MEMBANGUN PC *CLUSTER* DI LINGKUNGAN LABORATORIUM KOMPUTER TEKNIK ELEKTRO UIN SUSKA RIAU

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat

Untuk Memperoleh Gelar Sarjana Teknik Pada

Jurusan Teknik Elektro

Oleh :

AFIF ZULFIKAR
10555002373

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SULTAN SYARIF KASIM RIAU
PEKANBARU
2010

MEMBANGUN PC *CLUSTER* DI LINGKUNGAN LABORATORIUM KOMPUTER TEKNIK ELEKTRO UIN SUSKA RIAU

AFIF ZULFIKAR
NIM: 10555002373

Tanggal Sidang: 28 Januari 2010
Tanggal Wisuda: Februari 2010

Jurusan Teknik Elektro
Fakultas Sains dan Teknologi
Universitas Islam Negeri Sultan Syarif Kasim Riau
Jl. Soebrantas No. 155 Pekanbaru

ABSTRAK

Keberadaan komputer di laboratorium komputer Teknik Elektro UIN SUSKA Riau yang ada saat ini sudah mulai tertinggal oleh pesatnya perkembangan teknologi komputer. Permasalahan tersebut dapat diatasi dengan membangun komputer-komputer yang ada menjadi PC *cluster*. PC *cluster* yang dibangun pada penelitian ini terdiri dari 1 komputer *head node*, dan 9 komputer *compute node*. Topologi jaringan yang digunakan adalah topologi *hybrid* (*star* dan *tree*). Untuk sistem operasi yang digunakan dalam membangun *cluster* pada laboratorium komputer Teknik Elektro UIN SUSKA Riau adalah sistem operasi linux openSUSE 11.1, mekanisme pemrograman paralel yang digunakan adalah MPI, dan pustaka pemrograman paralel yang digunakan adalah MPICH1. PC *cluster* ini telah berhasil dibangun dan dapat menjalankan program paralel. Pengujian program paralel menggunakan program tes.c, yaitu program yang dibuat untuk menampilkan jumlah *node* yang digunakan. Disini hanya untuk membuktikan apakah *node cluster* dapat menjalankan program paralel secara bersamaan.

Kata Kunci : MPI, MPICH1, *node*, PC *cluster*, topologi *hybrid* (*star* dan *tree*)

BUILD PC CLUSTER IN ELECTRICAL ENGINEERING DEPARTEMENT COMPUTER LABORATORY UIN SUSKA RIAU ENVIRONMENT

AFIF ZULFIKAR
NIM : 10555002373

Date of Final Exam : January 28, 2010
Date of Graduation Cremony: February, 2010

Electrical Engineering Departement
Faculty of Sciences and Technology
State Islamic University of Sultan Syarif Kasim Riau
Soebrantas Street No. 155 Pekanbaru

ABSTRACT

Existence of computer in existing Electrical Engineering Departement State Islamic University of Sultan Syarif Kasim Riau computer laboratory in this time have started left behind by fast of technological growth of computer. The problems can overcome by developing existing computer become PC cluster. Woke up Cluster PC at this research consist of 1 head node computer, and 9 compute node computer. Network topology used at this research is hybrid (*star and tree*) *topology. For the operating system of which used in developing cluster at Electrical Engineering Departement State Islamic University of Sultan Syarif Kasim Riau computer laboratory is Linux OpenSuSE 11.1, parallel programming mechanism which used is MPI, and parallel programming libraries which used is MPICH1. This PC cluster have succeeded to woke up and can run parallel programming. Examination of parallel programming use tes.c program, this program made to present the amount of used node. At here just for proving do cluster node can run parallel program concurrently.*

Keywords : *hybrid star and tree topology, MPI, MPICH1, node, PC cluster*

DAFTAR ISI

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Keberadaan komputer di laboratorium komputer Teknik Elektro UIN SUSKA Riau yang ada saat ini sudah mulai tertinggal oleh pesatnya perkembangan teknologi komputer. Karena perkembangan teknologi komputer yang begitu pesat maka bila membeli komputer dengan kapasitas komputasi yang besar komputer tersebut juga akan cepat ketinggalan, dan biaya yang telah dikeluarkan akan terbuang percuma.

Pada awalnya proses komputasi hanya dapat dilakukan secara sekuensial saja. Sebuah prosesor hanya dapat melakukan sebuah proses saja dan proses yang lain harus menunggu untuk selanjutnya dapat dieksekusi oleh prosesor. Hal tersebut tentu membutuhkan waktu yang lama bagi prosesor untuk mengeksekusi sebuah proses atau melakukan proses komputasi. Akan tetapi seiring perkembangan teknologi mikroprosesor, proses komputasi kini dapat dilakukan secara paralel dengan menggunakan banyak prosesor untuk melakukan sebuah proses komputasi.

Perkembangan komputer *cluster* merupakan salah satu solusi untuk mengatasi hal tersebut. Sistem *cluster* dibuat untuk menciptakan suatu sistem komputer yang memiliki waktu pemrosesan yang sangat cepat, dan sumber daya yang sangat tinggi. Karakteristik utama pada sistem *cluster* adalah bagaimana suatu *task* dapat ditangani secara bersama-sama. Sistem *cluster* lebih difokuskan pada mekanisme sistem operasi dan *hardware* untuk menyatukan seluruh sumber daya pada setiap *node*, baik itu CPU, *memory*, *disk*, dan sebagainya. Untuk membangun *cluster* tersebut dibutuhkan paling sedikit dua komputer yang terhubung ke sebuah jaringan *private*.

Penelitian tentang PC *cluster* ini pernah dilakukan oleh Widyaputra (2008) dengan judul Perancangan *Cluster* Linux untuk Komputasi Paralel Octave. Dari hasil penelitian tersebut PC *cluster* yang dibangun menggunakan 3 unit komputer,

sistem operasi menggunakan distro Linux Fedora *Core* 6, pustaka pemrograman menggunakan LAM/MPI, dan topologi jaringan menggunakan topologi *star*.

Selanjutnya, penelitian yang dilakukan oleh Rianandra (2007) yang meneliti tentang Komputasi Paralel Penyelesaian Persamaan *Poisson* 1-D dengan *Massage Passing Interface* (MPI) pada Jaringan Komputer Berbasis Linux. Pada penelitiannya, PC *cluster* yang dibangun menggunakan 4 unit komputer, sistem operasi menggunakan distro SuSE Linux *Enterprise Desktop* 10, pustaka pemrograman menggunakan MPICH1, dan topologi jaringan menggunakan topologi *star*.

Bedanya dengan penelitian tersebut, pada penelitian ini peneliti menggunakan komputer laboratorium Teknik Elektro Universitas Islam Negeri Sultan Syarif Kasim Riau yang berjumlah 10 unit, dengan sistem operasi yang digunakan adalah distro Linux OpenSuSE 11.1. Selain distro ini bersifat gratis dan *open source*, distro Linux OpenSuSE 11.1 juga mudah untuk dikonfigurasikan karena terdapat YaST yang merupakan program bantu untuk memenuhi infrastruktur dari sebuah sistem *cluster*.

Pustaka pemrograman menggunakan MPICH1 (*Message Passing Interface Chameleon*1) yang merupakan implementasi dari MPI (*Message Passing Interface*). MPI adalah sebuah antarmuka pemrograman yang digunakan untuk mendistribusikan proses ke simpul-simpul komputasi lain dalam sebuah *cluster*. Pada penelitian ini topologi jaringan yang digunakan adalah topologi *hybrid* (*star* dan *tree*). Penggunaan topologi *hybrid* (*star* dan *tree*) pada penelitian ini dimaksudkan agar kedepannya dapat dimanfaatkan sebagai topologi dasar dalam membangun sistem *grid*. Topologi ini dipilih karena tingkat kerumitan instalasinya. Instalasi topologi *star* lebih mudah dibandingkan topologi *hybrid* (*star* dan *tree*).

Diharapkan dengan adanya komputer *cluster* di lingkungan laboratorium komputer Teknik Elektro Universitas Islam Negeri Sultan Syarif Kasim Riau dapat memberikan solusi terhadap permasalahan-permasalahan komputasi yang ada pada saat ini, dan dapat membantu mahasiswa maupun dosen yang membutuhkan sebuah sistem komputasi paralel.

1.2 Rumusan Masalah

Bagaimana membangun PC *cluster* di lingkungan laboratorium komputer Teknik Elektro UIN SUSKA Riau.

1.3 Batasan Masalah

Pada penelitian ini masalah yang diteliti dibatasi pada :

a) Sistem operasi yang digunakan adalah distro Linux OpenSuSE 11.1.
b) Komputer yang digunakan berjumlah 10 unit.
c) Topologi jaringan yang digunakan adalah topologi *hybrid* (*star* dan *tree*).
d) Pustaka pemrograman yang digunakan adalah MPICH1 dan bahasa pemrograman yang digunakan adalah bahasa C.
e) Komponen untuk membangun PC *cluster* adalah mesin komputasi, jaringan interkoneksi, dan perangkat lunak yang digunakan untuk mengembangkan aplikasi paralel.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah membangun PC *cluster* yang dapat memproses suatu *task* paralel secara simultan.

1.5 Metode Penelitian

Metode penelitian yang digunakan pada penelitian ini adalah sebagai berikut :

a. Metode studi literatur

   Metode ini digunakan untuk mengumpulkan informasi-informasi dan pengetahuan sebagai referensi dalam melakukan penelitian.

b. Instalasi perangkat keras

   Melakukan instalasi jaringan menggunakan topologi *hybrid* (*star* dan *tree*).

c. Instalasi perangkat lunak

   Melakukan instalasi sistem operasi distro Linux OpenSuSE 11.1, dan pustaka pemrograman MPICH1.

d. Pengujian

   Melakukan pengujian terhadap PC *cluster* yang telah dibangun.

1.6 Sistematika Penulisan

Laporan tugas akhir ini akan disusun secara sistematis dan dibagi menjadi lima bab dengan rincian sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, metode penelitian, dan sistematika penulisan laporan.

BAB II LANDASAN TEORI

Bab ini berisi tentang dasar teori yang digunakan dalam membangun dan menganalisa PC *cluster*.

BAB III PERANCANGAN PC *CLUSTER*

Bab ini berisi perancangan sistem *cluster* yang akan dikerjakan.

BAB IV PENGUJIAN DAN PEMBAHASAN

Bab ini berisi pengujian dan pembahasan sistem *cluster* yang telah dibangun.

BAB V KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan hasil tugas akhir dan saran-saran yang diperlukan untuk proses pengembangan selanjutnya.

BAB II
DASAR TEORI

2.1 *Parallel Processing*

Parallel processing adalah metode untuk memecahkan sebuah permasalahan yang besar dengan cara memecah permasalahan tersebut menjadi komponen-komponen yang lebih kecil dengan proses kalkulasi atau komputasi dilakukan secara paralel.

Komputasi paralel dapat digolongkan menurut tingkatan perangkat keras yang mendukung paralelisme yaitu:

a. *Multicore Computing*

   Multicore computing digolongkan dalam komputasi paralel karena *multicore* merupakan pendekatan perlahan menuju ke pemrosesan berorientasi paralel. *Multicore* prosesor adalah proses yang didalamnya terdapat *multiple execution unit* untuk tiap intinya. Prosesor ini berbeda dengan prosesor *superscalar*, yang dapat menangani *multiple* instruksi per siklus dari satu *thread*.

b. *Symmetric Multi Processing*

   Symmetric Multi Processing (SMP) adalah suatu sistem dengan lebih dari satu prosesor yang memiliki memori bersama. Semua prosesor dapat mengakses memori melalui bus. Bus bertanggung jawab mengatur permintaan pemakaian memori yang berlangsung secara simultan oleh beberapa prosesor. Bus juga bertanggung jawab untuk meyakinkan bahwa semua prosesor dilayani secara adil dengan waktu tunda akses yang minimum.

c. *Distributed Computing*

   Komputer terdistribusi atau biasa juga dikenal dengan *shared memory* prosesor merupakan sistem komputer dimana unsur-unsur pengolahan dihubungkan oleh suatu jaringan. *Distributed Computing* terbagi atas tiga kategori antara lain:

1. *Cluster Computing*

   Perbedaan utama untuk kategori ini dengan kategori lainnya adalah seberapa eratkah penggabungan antar *node*-nya. Komputasi membutuhkan komunikasi yang sering antar *node*. *Cluster* menggunakan sebuah jaringan terdedikasi yang sama, yang terletak di lokasi yang sangat berdekatan, dan bisa juga merupakan *node-node* yang bersifat homogen.

2. *Massive Parallel Processing*

   Massive Parallel Processing (MPP) adalah komputer tunggal dengan banyak prosesor terhubung melalui jaringan. MPP memiliki banyak kemiripan karakteristik dengan sistem *cluster*, tetapi MPP memiliki spesialisasi koneksi antar jaringan. MPP juga cenderung lebih besar dari pada *cluster* secara khas memiliki lebih dari 100 prosesor. Di dalam MPP, setiap CPU berisi memori sendiri, sistem operasi dan aplikasi. Setiap subsistem berkomunikasi dengan yang lainnya melalui interkoneksi berkecepatan tinggi.

3. *Grid Computing*

   Grid adalah *sharing* dan koordinasi yang dinamik, dapat tersebar secara geografis, memiliki sumber daya yang heterogen dan kesemuanya itu dapat diproses, menjalankan data dan melakukan penyimpanan data. Istilah heterogen diartikan sumber daya dapat berbeda di sistem operasi, arsitektur *hardware* maupun *software*, dan lokasi sumber daya dapat berjauhan satu sama lainnya. Istilah geografis dimaksudkan sumber daya berada ditempat yang berbeda baik secara institusi maupun secara pengaturan administrasi domain. Perbedaan pengaturan domain menjadi kendala terutama dalam pengaturan sumber daya (Fiade, 2008).

2.2 PC *Cluster*

PC *Cluster* adalah sebuah sistem komputer yang terdiri dari beberapa PC (*Personal Computer*) yang dihubungkan dalam satu jaringan untuk melakukan

sebuah pekerjaan komputasi ataupun simulasi secara bersama-sama. Setiap PC menjadi satu unit prosesor dengan masing-masing memiliki memori dari sebuah mesin komputasi paralel.

Pada sistem komputasi paralel terdiri dari beberapa unit prosesor dan beberapa unit memori. Ada dua teknik yang berbeda untuk mengakses data di unit memori, yaitu *shared memory address* dan *message passing*. Berdasarkan cara mengorganisasikan memori ini komputer paralel dibedakan menjadi *shared memory parallel machine* dan *distributed memory parallel machine*.

Di dalam sebuah *cluster* terdapat beberapa komputer yang disebut sebagai *node*. Masing-masing *node* tersebut merupakan mesin komputasi yang saling terintegrasi satu sama lain dengan sebuah jaringan. *Node-node* di dalam *cluster* tersebut dapat memiliki sistem operasi dan sistem *input* dan *output* masing-masing. Jika semua *node* di dalam *cluster* memiliki arsitektur dan sistem operasi yang sama, maka *cluster* tersebut dikatakan sebagai *cluster* yang homogen. Jaringan interkoneksi yang menghubungkan *node-node* di dalam *cluster* sebisa mungkin memiliki *bandwidth* yang tinggi dan latensi yang serendah mungkin karena kecepatan dan latensi di dalam jaringan *cluster* sangat menentukan performa komputasi yang dilakukan oleh *cluster*.

Untuk membangun sebuah *cluster* dibutuhkan empat buah komponen, yaitu dua buah komponen *hardware* dan dua buah komponen *software*. Komponen *hardware* yang pertama adalah *node* atau mesin komputasi dan yang kedua adalah jaringan interkoneksi, sedangkan komponen *software* yang pertama adalah *software* yang dapat digunakan oleh pengguna untuk mengembangkan aplikasi paralel dan yang kedua adalah *software* yang digunakan untuk memonitor dan mengatur aplikasi yang berjalan di dalam *cluster*.

PC *cluster* digunakan untuk melakukan komputasi dengan performa yang tinggi. Komputasi dengan performa tinggi tersebut dicapai dengan melakukan komputasi secara paralel yaitu menyelesaikan sebuah permasalahan komputasi dengan membagi permasalahan untuk diselesaikan secara simultan oleh masing-masing *node* di dalam *cluster* (Widyaputra, 2008).

Setiap *node* di dalam PC *cluster* yang dibangun di dalam penelitian ini adalah PC *cluster* yang menggunakan sistem operasi Linux OpenSuSE 11.1 dan perangkat-perangkat *input* dan *output* yang memiliki spesifikasi hamper sama setiap unitnya.

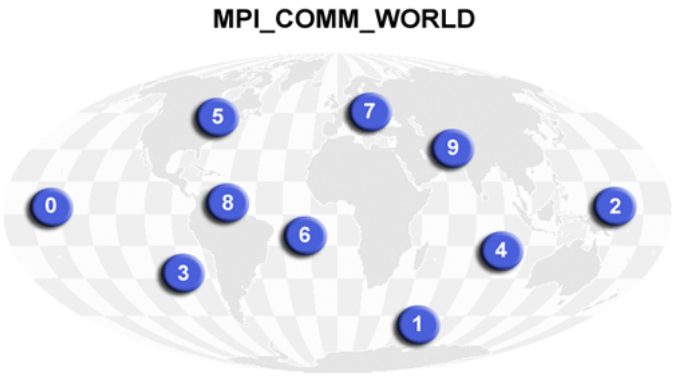
2.3 *Message Passing Interface* (MPI)

Model *message passing* telah dikenal luas sebagai salah satu model pemrograman aplikasi paralel. Dengan *message passing*, setiap proses yang terlibat dalam penyelesaian masalah secara paralel dapat berkomunikasi dengan cara mengirim pesan.

Konsep *message passing* kemudian dijadikan sebuah standar agar ada suatu cara yang sama dalam penulisan program. Standar ini dapat digunakan oleh berbagai macam pengguna dan dapat diimplementasikan secara efisien dalam berbagai macam komputer. Standar ini kemudian dikenal sebagai *Message Passing Interface* (MPI).

MPI berisi sekumpulan *Application Programming Interface* (API) yang dapat digunakan oleh pengembang aplikasi. API ini dirancang untuk dapat diterapkan pada bahasa pemrograman apa saja (*language independent*). MPI juga menetapkan protokol dan spesifikasi bagaimana setiap fitur dari MPI diimplementasikan.

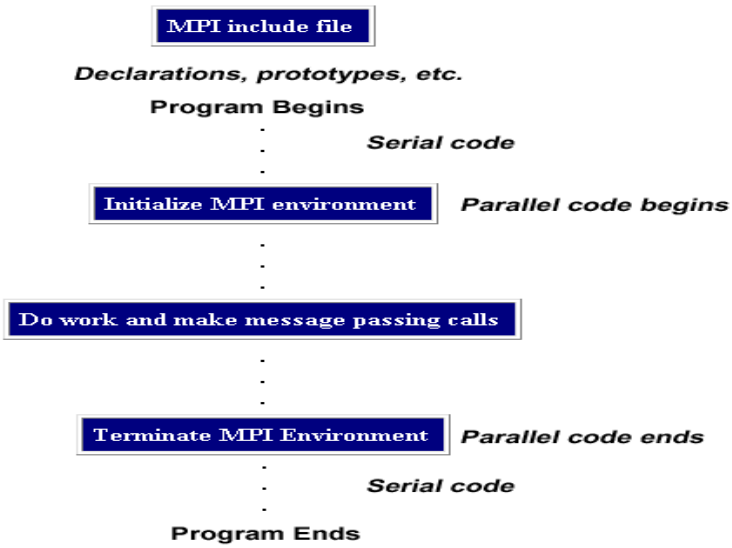
Gambar 2.1 menunjukkan sebuah himpunan komunikator MPI_COMM_WORLD dengan anggota *node-node* komputasi didalamnya. Hampir semua fungsi-fungsi atau *routine* menggunakan komunikator sebagai argumen. Salah satu komunikator yang paling sering digunakan adalah MPI_COMM_WORLD.



Gambar 2.1 Komunikator MPI_COMM_WORLD (Sumber : Barney, 2009)

Di dalam program MPI, harus terdapat pemanggilan fungsi MPI_Init sebagai inisialisasi program dan MPI_Finalize untuk terminasi program. Deklarasi variabel dan algoritma-algoritma yang berada diantara MPI_Init dan MPI_Finalize akan dijalankan secara paralel atau akan dibaca oleh semua proses yang berada di dalam komunikator.

Gambar 2.2 menunjukkan sebuah struktur program MPI. Penulisan program MPI untuk bahasa pemrograman seperti C, C++, dan Fortran selalu diawali dengan pemanggilan pustaka MPI (*MPI include file*). Setelah melakukan pemanggilan pustaka MPI, selanjutnya fungsi-fungsi MPI dapat dipanggil di dalam program. Fungsi yang harus dipanggil untuk menjalankan program MPI adalah fungsi yang digunakan untuk inisialisasi program MPI dan terminasi program MPI, yaitu MPI_Init dan MPI_Finalize. Program yang dituliskan diantara pemanggilan fungsi MPI_Init dan MPI_Finalize akan dieksekusi oleh semua *node* yang termasuk dalam himpunan komunikator MPI_COMM_WORLD.



Gambar 2.2 Struktur Program MPI (Sumber : Barney, 2009)

MPI adalah perangkat lunak yang memungkinkan sekumpulan komputer terlibat seperti satu sistem komputer paralel dan dapat digunakan sebagai sebuah sumber daya komputasi yang koheren. Komputer tersebut bisa berupa *workstation*, *multiprocessor*, *specialized grapich engine* sampai dengan *vector super computer* yang dihubungkan dengan jaringan. MPI memungkinkan eksekusi program pada setiap mesin dapat dikendalikan oleh *user* dan menjadi lingkungan komputasi yang *powerfull*. MPI digunakan untuk komputasi paralel dalam sistem yang terdistribusi. Pengguna MPI dapat menuliskan programnya dengan bahasa C, C++, FORTRAN77, dan FORTRAN90 untuk menjalankannya secara paralel dengan memanggil rutin *library* yang sesuai. Karena datanya terdistribusi, biasanya komputasi pada suatu proses akan membutuhkan suatu nilai data yang di *copy* dari proses lainnya. Oleh karena itu, bila proses A membutuhkan nilai data X yang disimpan pada memori di data B, maka program tersebut harus meng-*input*-kan suatu baris seperti :

```
if (I am processor A) then
call MPI_Send (X)
else if (I am processor B) then
call MPI_Recev (X)
end
```

Jelas bahwa untuk mengeksekusi sebuah program paralel yang menggunakan MPI akan terlihat berbeda dengan bila menggunakan versi sekuensial. Pengguna harus membagi data masalah ke beberapa proses, menulis ulang algoritma untuk membagi kerja ke beberapa proses dan menambah beberapa program untuk mengirimkan nilai yang dibutuhkan dari sebuah proses dimana data tersebut berada ke sebuah proses yang membutuhkan nilai tersebut.

MPI dikembangkan pada tahun 1993-1994 oleh sekelompok peneliti yang berasal dari kalangan pemerintahan, industri dan akademia. MPI merupakan salah satu standar pertama untuk menjalankan program paralel prosesor, dan yang merupakan pelopor dalam *basic message passing*.

2.3.1 Fungsi-fungsi Dasar MPI

Fungsi-fungsi dasar MPI cukup sederhana dan meliputi fungsi inisialisasi dan fungsi dasar *send*/*receive*, untuk lengkapnya dapat dilihat pada tabel 2.1.

Tabel 2.1 Daftar fungsi-fungsi utama pada MPI (Sumber : Barney, 2009)

Nama Fungsi	**Tujuan Fungsi**	***Syntax***
MPI_Init	Inisialisasi lingkungan MPI. Fungsi ini hanya boleh dipanggil sekali, sebelum penggunaan fungsi-fungsi MPI yang lain.	**`MPI_Init (&argc,&argv)`** **`MPI_INIT (ierr)`**

Tabel 2.1 Lanjutan

Nama Fungsi	**Tujuan Fungsi**	***Syntax***
MPI_Comm_size	Untuk menentukan jumlah prosesor yang tergabung dalam sebuah *communicator* (contohnya MPI_COMM_WORLD)	**MPI_Comm_size (comm,&size)** **MPI_COMM_SIZE (comm,size,ierr)**
MPI_Comm_ rank	Untuk menentukan *rank* dari prosesor yang memanggil fungsi ini dalam sebuah *communicator* (contohnya MPI_COMM_WORLD)	**MPI_Comm_rank (comm,&rank)** **MPI_COMM_RANK (comm,rank,ierr)**
MPI_Abort	Untuk menghentikan semua proses MPI yang berjalan dalam *communicator*	**MPI_Abort (comm,errorcode)** **MPI_ABORT (comm,errorcode,ierr)**
MPI_Get_ processor_ name	Untuk Mendapatkan nama prosesor tiap *node*	**MPI_Get_processor_name (&name,&resultlength)** **MPI_GET_PROCESSOR_NAME (name,resultlength,ierr)**
MPI_ Initialized	Menunjukkan apakah telah dipanggil MPI_Init	**MPI_Initialized (&flag)** **MPI_INITIALIZED (flag,ierr)**
MPI_Wtime	Untuk melihat waktu yang telah berjalan semenjak panggilan terakhir terhadap fungsi ini sebelumnya	**MPI_Wtime ()** **MPI_WTIME ()**
MPI_Wtick	Untuk melihat resolusi dalam detik dari MPI_Wtime	**MPI_Wtick ()** **MPI_WTICK ()**

Tabel 2.1 Lanjutan

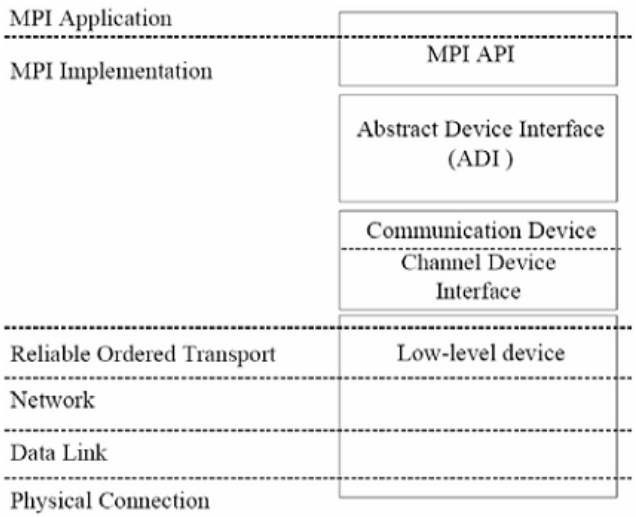
Nama Fungsi	Tujuan Fungsi	*Syntax*
MPI_Finalize	Untuk menghentikan lingkungan eksekusi MPI pada *communicator*	`MPI_Finalize ()` `MPI_FINALIZE (ierr)`
MPI_Send	Fungsi dasar untuk mengirim data	`MPI_Send (&buf,count,datatype,dest,tag,comm)` `MPI_SEND (buf,count,datatype,dest,tag,comm,ierr)`
MPI_Recv	Fungsi dasar untuk memerintahkan komputer agar menunggu data yang akan masuk	`MPI_Recv (&buf,count,datatype,source,tag,comm,&status)` `MPI_RECV (buf,count,datatype,source,tag,comm,status,ierr)`
MPI_Bcast	Mem-*broadcast* sebuah pesan ke semua prosesor yang tergabung dalam sebuah *communicator* (contohnya MPI_COMM_WORLD)	`MPI_Bcast (&buffer,count,datatype,root,comm)` `MPI_BCAST (buffer,count,datatype,root,comm,ierr)`

2.3.2 *Message Passing Interface Chameleon* 1 (MPICH1)

MPICH1 adalah implementasi *portable* dari MPI, standar untuk *message passing libraries*, yang tersedia secara *free*. MPICH1 dikembangkan oleh *Matematics and Computer Science Division* pada laboratorium nasional Argonne. MPI sendiri merupakan singkatan dari *Message Passing Interface* sedangkan kata CH berasal dari *Chameleon*.

Arsitektur MPICH ditunjukkan pada Gambar 2.3. Abstraksi program aplikasi yang dibuat oleh *user* dinyatakan pada lapisan API. Fungsi-fungsi pustaka yang tersedia pada MPI dinyatakan dalam *header* mpi.h. Pengaktifan MPI dimulai dengan menjalankan perintah MPI_Init(&argc, &argv); pada program utama, dilanjutkan dengan menentukan *ranking* dari tiap *node* yang

menjalankan program aplikasi dengan perintah MPI_Comm_rank (MPI_COMM_WORLD, &my_rank); my_rank adalah bilangan bulat positif, bernilai nol berarti program berjalan pada komputer *master*, sebaliknya bernilai tidak sama dengan nol berarti program berjalan pada komputer *slave*. MPI_COMM_WORLD adalah konstanta yang telah terdefinisi untuk mengendalikan proses-proses yang ada pada saat MPI dimulai.



Gambar 2.3 Arsitektur MPICH (Sumber : Suhartanto, 2006)

Untuk mengetahui jumlah prosesor (*node*) yang aktif digunakan perintah MPI_Comm_size(MPI_COMM_WORLD, &p). Komunikasi dilakukan secara berurutan dari lapisan teratas sampai ke lapisan fisik, pada sisi penerima berlaku sebaliknya, yaitu dari lapisan fisik ke atas. Misalnya ada perintah MPI_Send(*message*, strlen(*message*)+1, MPI_CHAR,dest, tag, MPI_COMM_WORLD); maka pesan (*message*) sepanjang strlen (*message*)+1 dan bertipe karakter (MPI_CHAR) dikirim ke prosesor tujuan (*dest*). Pada lapisan ADI pesan tersebut diterima oleh Send_handle yaitu pengendali pengiriman data pada lapisan ADI, kemudian pada lapisan *Channel Device* paket tersebut diterima dengan pengendali MPID_SendControl dan MPID_SendChannel. Selanjutnya, pada lapisan *low-level device* komunikasi dilakukan dengan protokol yang tersedia.

Misalnya MPICH berjalan pada Microsoft Windows 2000, maka protokol yang digunakan adalah TCP/IP. Pada sisi penerima, lapisan komunikasi atau *channel device interface* memiliki pengendali MPID_ControlMsgAvail dan MPID_RecvAnyControl. Kedua pengendali ini meneruskan paket ke lapisan ADI. Pada lapisan ini, terdapat dua pengendali, yaitu: PostedRecv_Handles dan UnexpectedRecv_Handles, masing-masing digunakan untuk mengetahui pengiriman paket yang terkirim dengan benar dan yang salah. Selanjutnya, pada lapisan aplikasi, data diterima dengan perintah MPI_Recv(message, len, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status); dan untuk mengakhiri MPI digunakan perintah MPI_Finalize();.

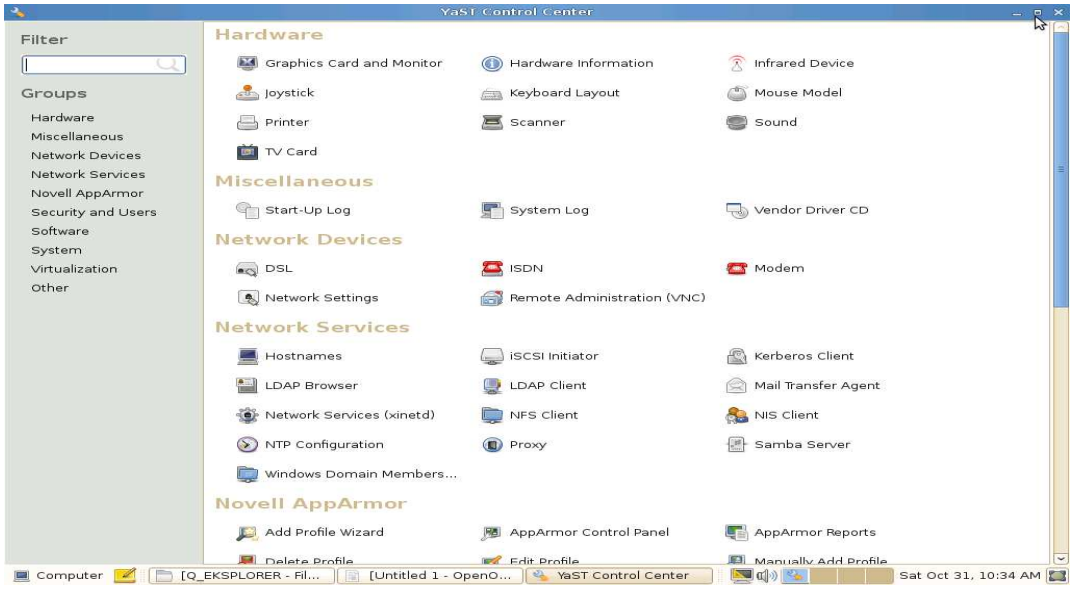
2.4 OpenSuSE 11.1

SUSE adalah singkatan Jerman dari *Software und System Entwicklung* atau *Software and System Development*. SUSE didirikan pada tanggal 2 September 1992 dengan pendirinya adalah Roland Dyroff, Thomas Fehr, Burchard Steinbild, dan Hubert Mantel. OpenSUSE adalah distro Linux versi komunitas yang didukung dan disponsori oleh Novell. OpenSuSE merupakan distro linux *open source* dan gratis yang menjadi dasar pengembangan bagi distro linux komersil yang disediakan oleh Novell, SUSE Linux *Enterprise Server* (SLES) dan SUSE Linux *Enterprise Desktop* (SLED).

Salah satu keunggulan utama dari OpenSuSE dibandingkan distro Linux lainnya adalah kelengkapan pustaka dan berlimpahnya *software* yang disertakan. Bersama Red Hat, SuSE adalah distro Linux versi awal yang terus bertahan dan berkembang hingga sekarang. Pada OpenSuSE terdapat YaST (*Yet another System Tools*) yang merupakan ciri khas dari OpenSUSE. YaST adalah *tools* untuk me-*maintenance* dan me-*manage* penggunaan *software* dan *hardware* di OpenSuSE.

YaST merupakan program bantu untuk mengkonfigurasikan Linux dengan menu interaktif yang mudah dimengerti. YaST juga dirancang dengan abstarksi *layer* antara fungsi dan antarmuka pengguna. Artinya kita dapat menjalankan modul YaST dalam modul *text console* atau dalam modul grafis dan kita akan

mendapatkan fungsionalitas dan menu yang sama. Modul YaST dapat juga dipanggil dari GNOME *control center*, dimana *icon* untuk modul YaST telah digabungkan dengan *icon* GNOME yang spesifik. Dari waktu ke waktu jumlah modul yang tersedia pada YaST secara terus-menerus meningkat, dan kemampuan yang ditawarkan juga telah ditingkatkan.



Gambar 2.4 YaST *control center*

Seperti yang dapat kita lihat, kita dapat mengatur sistem Linux melalui YaST tanpa harus menyentuh konfigurasi *file*. Ini adalah warisan dari pengembang SuSE agar SuSE mudah untuk di konfigurasi. YaST mempunyai struktur menu konfigurasi:

1. ***General help for installation***

   File help yang berisi tentang tombol-tombol yang dipergunakan untuk navigasi YaST.

2. ***Adjustments of Installation***

   Bagian untuk instalasi atau *update* instalasi.

   a. *Select Language*
   b. *Select Keymap*
   c. *Select Installation Medium*
   d. *Configure harddisk partitions*
   e. *Set target partitions/filesystems*

f. *Installation to a directory*

3. *Choose/Install Packages*

Penambahan atau pengurangan (*uninstall*) paket aplikasi.

4. *Update System*

Pengecekan terhadap perubahan program-program lain serta sistem yang terpengaruh dengan instalasi program yang baru.

5. *System Administration*

Konfigurasi sistem, termasuk *hardware, network* dan *user*. Bagian ini terdiri dari:

a. *Integrate hardware into system*
   i. *Mouse configuration*
   ii. *Modem*
   iii. *CD-ROM*
   iv. *Configure printers*
   v. *Configure ISDN hardware*
   vi. *Configure your scanner*
   vii. *Configure networking device*

b. *Kernel and boot configuration*
   i. *Select boot kernel*
   ii. *Create a boot disk*
   iii. *Create a rescue disk*
   iv. *LILO Configuration*

c. *Network configuration*

d. *Network base configuration*
   i. *Change hostname*
   ii. *Configure network services*
   iii. *Configure name server*
   iv. *Configure YP Client*
   v. *DHCP client*
   vi. *Configure sendmail*
   vii. *Configure ISDN parameters*

viii. *Configure a PPP network*

ix. *Administer remote printers*

x. *Connect to printer via Samba*

xi. *Connect to printer via novell network*

e. *Login configuration*

i. *Settings of SuSE* WM

ii. *User Administration*

iii. *Group Administration*

iv. *Create backups*

v. *Set the console font*

vi. *Set Time Zone*

f. *Configure XFREE86 (tm)*

g. *Configure GPM*

h. *Security Settings*

i. *Change Configuration file*

6. *Show README File For Installation Media*

Menampilkan petunjuk untuk media instalasi.

7. *CopyRight*

Hak cipta dari OpenSuSE.

8. *Exit YaST*

Keluar dari YaST.

2.5 Topologi Jaringan

Jaringan Komputer dapat diartikan sebagai suatu himpunan interkoneksi sejumlah komputer. Dua buah komputer dikatakan membentuk suatu *network* atau jaringan komputer bila keduanya dapat saling bertukar informasi. Secara umum, jaringan mempunyai beberapa manfaat yang lebih dibandingkan dengan komputer yang berdiri sendiri (*stand alone*), yaitu dalam hal:

1. Jaringan memungkinkan manajemen sumber daya lebih efisien. Misalnya, banyak pengguna dapat saling berbagi *printer* tunggal dengan kualitas tinggi, dibandingkan memakai *printer* kualitas rendah di masing-masing

meja kerja. Selain itu, lisensi perangkat lunak jaringan dapat lebih murah dibandingkan lisensi *stand alone* terpisah untuk jumlah pengguna sama.

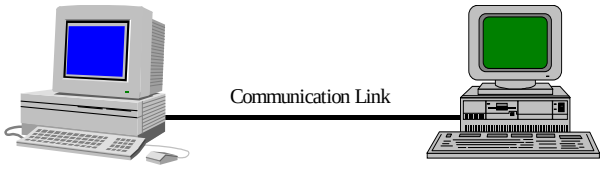
2. Jaringan membantu mempertahankan informasi agar tetap handal dan *up to date*. Sistem penyimpanan data terpusat yang dikelola dengan baik memungkinkan banyak pengguna mengakses data dari berbagai lokasi yang berbeda, dan membatasi akses ke data sewaktu sedang diproses.
3. Jaringan membantu mempercepat proses berbagi data (*data sharing*). *Transfer data* pada jaringan selalu lebih cepat dibandingkan sarana berbagi data lainnya yang bukan jaringan (*flashdisk*, *disket*, CD, dan lain sebagainya).
4. Jaringan memungkinkan kelompok kerja berkomunikasi dengan lebih efisien. Surat dan penyampaian pesan elektronik (*e-mail*) merupakan substansi sebagian besar sistem jaringan, disamping sistem penjadwalan, pemantauan proyek, konferensi *online* dan *groupware*, dimana semuanya membantu tim bekerja lebih produktif.
5. Jaringan membantu usaha dalam melayani *client* mereka secara lebih efektif. Akses jarak jauh ke data terpusat memungkinkan karyawan dapat melayani *client* dilapangan dan *client* dapat langsung berkomunikasi dengan pemasok.

Topologi jaringan komputer adalah suatu cara menghubungkan komputer yang satu dengan komputer lainnya sehingga membentuk jaringan. Dalam suatu jaringan komputer jenis topologi yang dipilih akan mempengaruhi kecepatan komunikasi. Untuk itu maka perlu dicermati kelebihan/keuntungan dan kekurangan/kerugian dari masing-masing topologi berdasarkan karakteristiknya.

A. *Point to point*

Jaringan *point to point* merupakan jaringan kerja yang paling sederhana tetapi dapat digunakan secara luas. Begitu sederhananya jaringan ini, sehingga seringkali tidak dianggap sebagai suatu jaringan tetapi hanya merupakan jalur komunikasi biasa. Pada jenis topologi ini, kedua *node* mempunyai kedudukan setingkat, sehingga *node* manapun dapat memulai

dan mengendalikan hubungan dalam jaringan. Data dikirim dari satu *node* langsung ke *node* lainnya sebagai penerima.



Gambar 2.5 *Point to Point* (Sumber : Aryanto, 2008)

B. Topologi *Star*

Topologi *star* adalah topologi jaringan dimana setiap *node* dalam jaringan terhubung dengan *node* pusat dengan hubungan *point to point*. Semua data yang ditransmisikan ke-*node* dalam jaringan selalu ditransmisikan ke-*node* pusat yang kemudian ditransmisikan ke-*nodes* di dalam jaringan, walaupun *node* pusat mungkin juga sebuah titik koneksi biasa tanpa ada perangkat aktif untuk mengulang sinyal. Sebuah koneksi *point to point* kadang dikategorikan sebagai bagian khusus dari topologi *star*. Maka dari itu jenis jaringan terkecil dari topologi *star network* akan terdiri dari sebuah koneksi *point to point* ke-*node* kedua yang diatur oleh *hub*/*switch*. Berdasarkan hal tersebut, tipe jaringan terkecil berikutnya dari topologi *star network* terdiri dari satu *node* pusat yaitu *hub* dengan dua koneksi yang terpisah kedua *node* cabang. Walaupun kebanyakan jaringan yang didasarkan pada topologi ini memerlukan penggunaan *hub* sebagai *node* pusat, namun masih ada kemungkinan untuk mengimplementasikan sebuah jaringan yang didasarkan pada topologi *star* dengan menggunakan sebuah komputer atau bahkan titik koneksi biasa sebagai *hub* atau *node* pusat. Topologi ini dirancang dengan setiap *node* (*file server*, *workstation*, dan *peripheral*) terhubung secara langsung ke jaringan pusat atau biasa disebut konsentrator. Data pada sebuah jaringan *star* selalu melalui *hub* atau konsentrator sebelum menuju sasaran. *Hub* atau konsentrator mengatur dan mengelola seluruh jaringan. Selain itu, *hub* juga dapat berperan sebagai *repeater* untuk data *flow*.

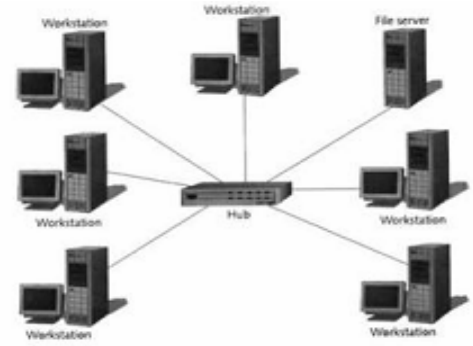
Konfigurasi semacam ini biasanya memakai kabel *twisted pair*. Selain itu bisa juga memakai kabel *coaxial* ataupun kabel serat optik.

Kelebihan topologi *star*:

- Jaringan tidak mudah terganggu oleh adanya koneksi baru maupun saat adanya komputer yang tidak disambung.
- Mudah mendeteksi gangguan pada jaringan.
- Mudah pengaplikasiannya.

Kekurangan topologi *star*:

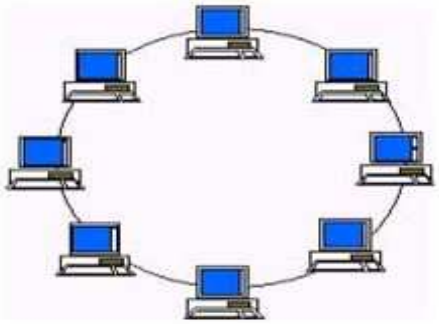
- Memerlukan kabel yang cukup panjang.
- Jika *hub*/konsentrator gagal berfungsi maka semua jaringan akan terputus.



Gambar 2.6 Topologi *Star* (Sumber : Supriadi, 2007)

C. Topologi *Ring*

Metode *ring* menghubungkan komputer sehingga berbentuk *ring* (lingkaran). Setiap simpul mempunyai tingkatan yang sama. Jaringan akan disebut sebagai *loop*, data dikirimkan kesetiap simpul dan setiap informasi yang diterima simpul diperiksa alamatnya apakah data itu untuknya atau bukan.



Gambar 2.7 Topologi *Ring* (Sumber : Kurniawan, 2009)

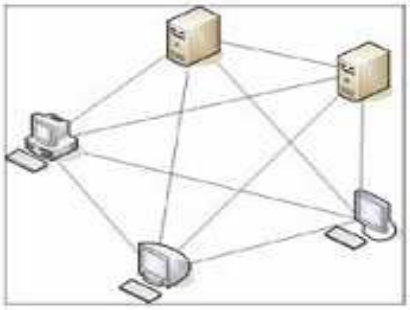
D. Topologi *Bus*

Media penghantar untuk jenis topologi *bus* adalah kabel Koaksial. Topologi *bus* menggunakan metode *unicast, multicast* dan *broadcast*. *Unicast* adalah komunikasi antar satu pengirim dengan satu penerima di jaringan. *Multicast* adalah komunikasi antara satu pengirim dengan banyak penerima di jaringan. Sedangkan pada *broadcast*, setiap titik akan menerima dan menyimpan *frame* yang disalurkan/dihantarkan.

Gambar 2.8 Topologi *Bus* (Sumber : Supriadi, 2007**)**

E. Topologi *Mesh*

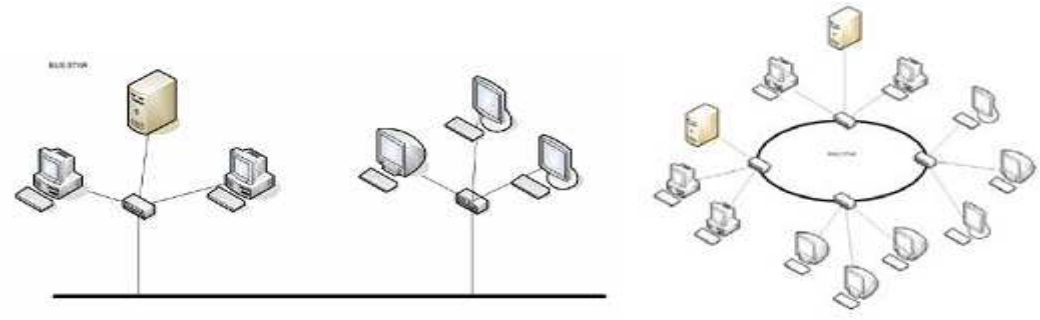
Topologi *mesh* dibangun dengan memasang banyak *link* pada setiap komputer. Hal ini dimungkinkan karena pada setiap komputer terdapat lebih dari satu NIC. Topologi ini secara teori memungkinkan akan tetapi tidak praktis dan biayanya cukup tinggi. Topologi *mesh* memiliki tingkat *redundancy* yang tinggi.



Gambar 2.9 Topologi *Mesh* (Sumber : Supriadi, 2007)

F. Topologi *Hybrid*

Topologi *hybrid* adalah jaringan yang dibentuk dari berbagai topologi dan teknologi. Sebuah topologi *hybrid* memiliki semua karakterisitik dari topologi dasar yang terdapat dalam jaringan tersebut.

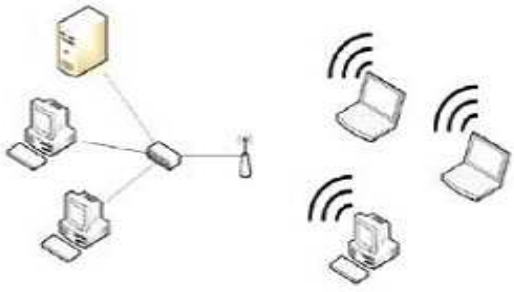


(a) (b)

Gambar 2.10 Topologi *Hybrid.* (a) *bus-star*, (b) *ring-star* (Sumber : Supriadi, 2007)

G. Topologi *Wireless*

Topologi wireless menggunakan gelombang radio untuk berkomunikasi dengan yang lainnya. Topologi *wireles* ini merupakan topologi yang sedang *trend* saat ini, karena mempunyai keunggulan lebih *mobile* dalam berkomunikasi. Topologi ini dapat berdiri sendiri dan secara umum banyak dipadukan dengan topologi dasar dalam aplikasinya.

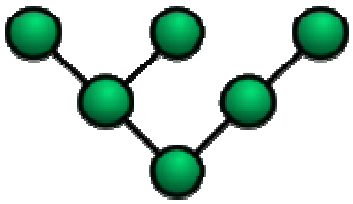


Gambar 2.11 Topologi *Wireless* (Sumber : Supriadi, 2007)

H. Topologi *Tree*

Topologi Jaringan Pohon (*tree*) disebut juga sebagai topologi jaringan bertingkat. Topologi ini biasanya digunakan untuk interkoneksi antar sentral dengan hirarki yang berbeda. Untuk hirarki yang lebih rendah digambarkan

pada lokasi yang rendah dan semakin keatas mempunyai hirarki semakin tinggi. Topologi jaringan jenis ini cocok digunakan pada sistem jaringan komputer.



Gambar 2.12 Topologi *Tree* (Sumber : Aryanto, 2008)

Pada jaringan pohon, terdapat beberapa tingkatan simpul (*node*). Pusat atau simpul yang lebih tinggi tingkatannya, dapat mengatur simpul lain yang lebih rendah tingkatannya. Data yang dikirim perlu melalui simpul pusat terlebih dahulu. Misalnya untuk bergerak dari komputer dengan *node*-3 ke komputer *node*-7 seperti halnya pada gambar, data yang ada harus melewati *node*-3, 5 dan *node*-6 sebelum berakhir pada *node*-7. Keungguluan jaringan model pohon seperti ini adalah, dapat terbentuknya suatu kelompok yang dibutuhkan setiap saat. Sebagai contoh, perusahaan dapat membentuk kelompok yang terdiri atas terminal pembukuan, serta pada kelompok lain dibentuk untuk terminal penjualan. Adapun kelemahannya adalah, apabila simpul yang lebih tinggi kemudian tidak berfungsi, maka kelompok lainnya yang berada dibawahnya akhirnya juga menjadi tidak efektif. Cara kerja jaringan pohon ini relatif menjadi lambat.

BAB III
PERANCANGAN PC *CLUSTER*

3.1 Analisis Kebutuhan Sistem *Cluster*

Cluster yang dibangun dalam penelitian ini adalah *cluster* yang didedikasikan untuk komputasi paralel. *Cluster* tersebut dibangun secara homogen yaitu dengan menyeragamkan sistem operasi dan menggunakan perangkat keras yang tidak jauh berbeda pada masing-masing komputer.

Untuk membangun sebuah *cluster* yang didedikasikan untuk komputasi paralel diperlukan beberapa persyaratan yang ditunjukkan oleh poin-poin sebagai berikut:

a. *Cluster* harus memiliki kemampuan mendistribusikan proses komputasi dari mesin *head node* ke mesin-mesin *compute node* yang lain.
b. *Head node* harus dapat berkomunikasi dengan mesin-mesin *compute node* tanpa harus melakukan proses otentikasi.

Dari persyaratan-persyaratan yang ditunjukkan oleh poin-poin di atas, dapat diambil beberapa solusi sebagai berikut :

a) Pendistribusian proses komputasi menggunakan MPICH1. MPICH1 adalah aplikasi yang mengimplementasikan MPI. MPI dipilih karena cocok untuk sistem yang homogen dibandingkan dengan PVM (*Parallel Virtual Machine*) yang lebih cocok untuk sistem yang heterogen. (Suhartanto, 2006).
b) Proses komunikasi antara *head node* dengan *node* dilakukan dengan *Remote Shell* (RSH). Ada banyak sistem untuk berkomunikasi antar *node* seperti rlogin, telnet, RSH, dan SSH (*Secure Shell*). RSH dipilih karena mudah dikonfigurasi dibandingkan telnet. SSH adalah aplikasi yang jauh lebih aman dikarenakan menggunakan jalur *secure*, akan tetapi SSH juga membutuhkan spesifikasi komputer yang tinggi. Karena PC *cluster* yang dibangun pada penelitian ini tidak terhubung ke layanan internet, peneliti menganggap aplikasi RSH lebih cocok untuk

diimplementasikan karena tidak membutuhkan tingkat keamanan yang tinggi.

Tabel 3.1 menunjukkan ringkasan persyaratan yang harus dipenuhi untuk membangun sebuah *cluster* dengan solusi yang dapat diambil untuk memenuhi persyaratan tersebut.

Tabel 3.1 Persyaratan dan Solusi dalam Membangun *Cluster*

Persyaratan	Solusi
Kemampuan dalam mendistribusikan proses komputasi	Menggunakan MPICH
Sistem komunikasi antar *node* yang saling percaya	Menggunakan RSH

Sistem operasi yang digunakan adalah sistem operasi distro Linux OpenSuSE 11.1, karena sistem operasi tersebut dianggap mudah untuk mengimplementasikan solusi-solusi yang ditunjukkan oleh poin-poin diatas. Linux juga merupakan sistem operasi yang bersifat *open source* sehingga dapat digunakan secara bebas dan gratis.

3.1.1 *Head Node*

Head node merupakan komputer di dalam sebuah *cluster* yang digunakan untuk memberikan *tasks* yang akan dikerjakan oleh *compute node*. Komputer ini juga digunakan sebagai tempat bagi pengguna *cluster* untuk mengakses *cluster* sehingga pengguna dapat memberikan tugas-tugas komputasi ke dalam *cluster*.

Head node harus dapat diakses melalui jaringan, karena *head node* merupakan tempat akses dan memberikan *tasks* ke *compute node*. *Head node* menggunakan perangkat keras sebagai berikut :

a) *Hard disk* dengan kapasitas 80 gigabyte.
b) *RAM* (*Random Access Memory*) dengan kapasitas 256 megabyte.
c) Sebuah *ethernet card* 100 Mbps
d) CPU Intel Pentium 4 2,4 GHz.

3.1.2 *Compute Node*

Compute node adalah komputer pekerja yang menerima *task* dari *head node* dan memproses *task* tersebut. Komputer yang difungsikan sebagai *compute node* hanya dapat diakses oleh *head node* untuk melakukan proses komputasi. *Compute node* menggunakan perangkat keras sebagai berikut :

a) *Hard disk* dengan kapasitas 40 gigabyte.
b) *RAM* (*Random Access Memory*) dengan kapasitas 256 megabyte.
c) Sebuah *ethernet card* 100 Mbps.
d) CPU Intel Pentium4 2,4 GHz.

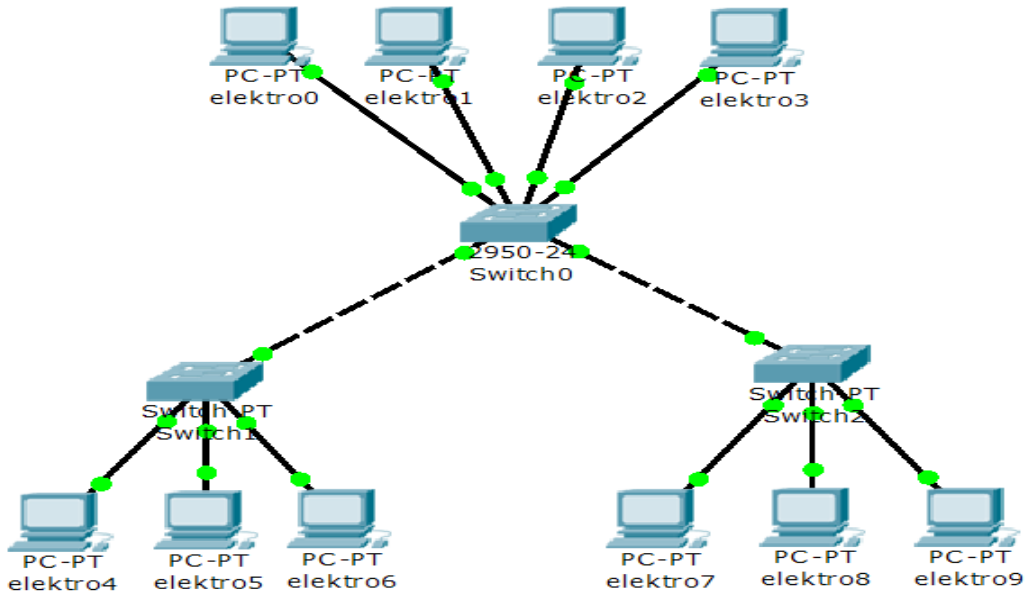
Untuk membangun sebuah *cluster*, diperlukan langkah-langkah yang sistematis dan berurutan. Langkah-langkah tersebut ditunjukkan oleh Gambar 3.1.

Gambar 3.1 Langkah-langkah Membangun *Cluster*

3.2 Instalasi Jaringan *Cluster*

Topologi yang digunakan pada *cluster* ini adalah topologi *hybrid* (*star* dan *tree*). Topologi ini dipilih karena tingkat kerumitan instalasinya. Instalasi topologi *star* lebih mudah dibandingkan topologi *hybrid*. Penggunaan topologi *hybrid* (*star*

dan *tree*) pada penelitian ini dimaksudkan agar kedepannya dapat dimanfaatkan sebagai topologi dasar dalam membangun sistem *grid*. Gambar 3.2 menunjukkan komputer 1 *head node* dan 9 *compute node* yang saling terhubung membentuk sebuah jaringan dengan topologi *hybrid*.



Gambar 3.2 Topologi Jaringan *Cluster*

Masing-masing *node* diberi sebuah IP *address private* kelas C (192.168.1.1 – 192.168.1.255). Komputer *head node* diberi IP *address* 192.168.1.1, komputer *compute node* 1 diberi IP *address* 192.168.1.2, komputer *compute node* 2 diberi IP *address* 192.168.1.3, dan seterusnya sampai pada komputer *compute node 9* diberi IP *address* 192.168.1.10.

Diperlukan pula sistem penamaan *hostname* untuk *node-node* sehingga mempermudah dalam mengingat nama *node-node* di dalam *cluster*. *Node-node cluster* yang dibangun dalam penelitian ini diberi nama elektro0 untuk *head node*, elektro1 untuk *compute node* 1, elektro2 untuk *compute node* 2, dan seterusnya sampai pada elektro9 untuk *compute node* 9.

Tabel 3.2 Daftar *Hostname*, IP *Address*, dan Fungsi Anggota *Cluster*.

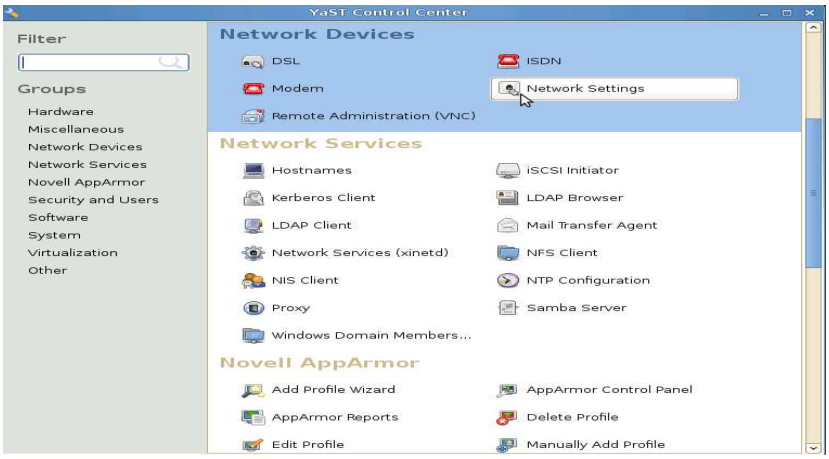
Hostname	**IP *Address***	**Fungsi**
elektro0	192.168.1.1	*Head node*
elektro1	192.168.1.2	*Compute node*
elektro2	192.168.1.3	*Compute node*
elektro3	192.168.1.4	*Compute node*
elektro4	192.168.1.5	*Compute node*
elektro5	192.168.1.6	*Compute node*
elektro6	192.168.1.7	*Compute node*
elektro7	192.168.1.8	*Compute node*
elektro8	192.168.1.9	*Compute node*
elektro9	192.168.1.10	*Compute node*

Untuk melakukan konfigurasi IP *address* dan pemberian *hostname* pada OpenSuSE 11.1, dapat dilakukan langkah berikut:

1. Dipilih menu *Computer* | *System* | YaST.
2. Tampil halaman konfirmasi untuk memasukkan *password*. Dimasukkan *password* administrator dengan *password* : “teritory”.

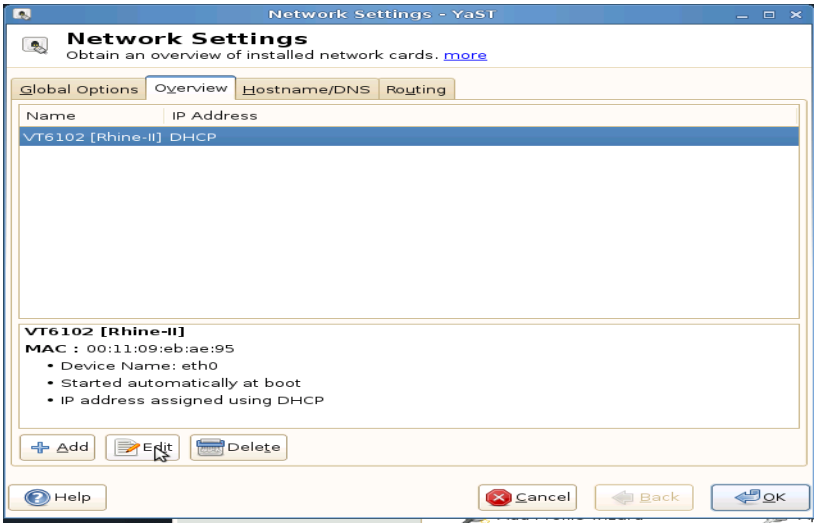
Gambar 3.3 Konfirmasi *Password* Administrator

3. Dipilih menu *Network Device* | *Network Settings*.



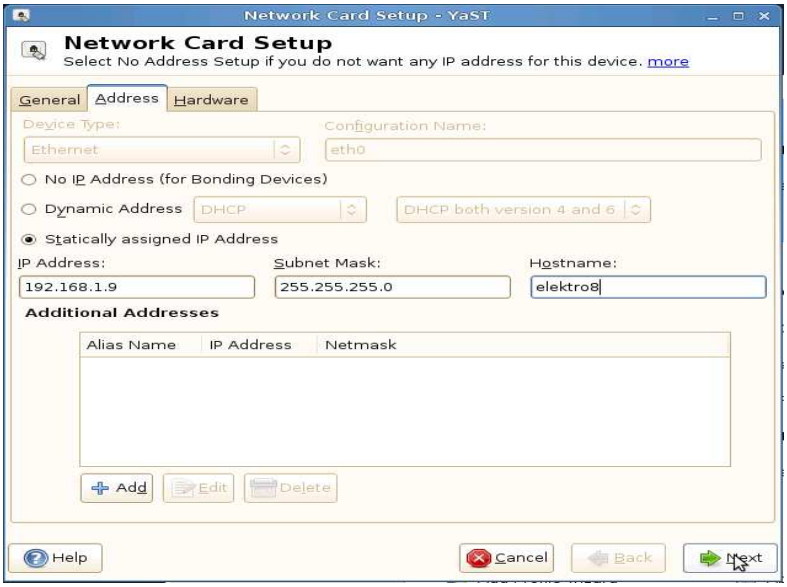
Gambar 3.4 Menu YaST *Network Devices*

4. Pada halaman *Network Settings*, dipilih kartu jaringan yang ingin dikonfigurasi, lalu diklik *edit*.



Gambar 3.5 Halaman *Network Settings*

5. Selanjutnya dipilih *option Statically assigned* IP *Address* untuk IP *Address Static*. Pada halaman ini juga nama *hostname* dimasukkan. Diklik *next* untuk mengaktifkan perubahan.



Gambar 3.6 Pemberian IP *Address Static* dan *Hostname*

3.3 Instalasi dan Konfigurasi *Remote Shell* (RSH)

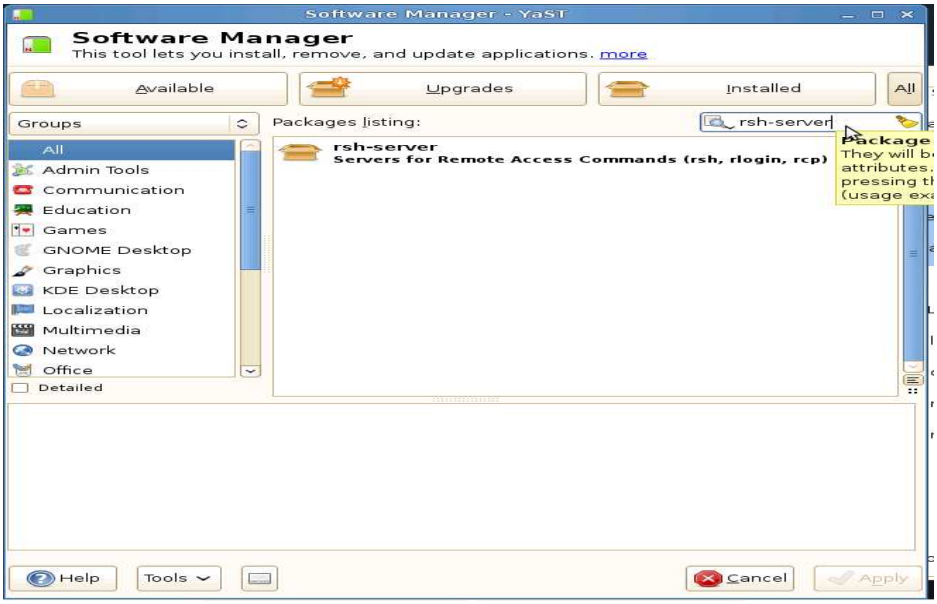
RSH dalam *cluster* digunakan untuk melakukan komunikasi antar *node*. MPI mengharuskan *head node* untuk mengeksekusi aplikasi-aplikasi paralel di setiap *node*. Dengan RSH hal tersebut dapat dilakukan karena RSH adalah aplikasi yang digunakan dalam sistem UNIX atau variannya untuk melakukan pengaksesan ke sebuah mesin secara *remote*.

Remote shell dipasang dalam *cluster* sebagai *service* yang digunakan oleh MPICH untuk berkomunikasi antar *node*. Dengan menggunakan RSH, *login* ke *remote host* dapat dilakukan tanpa *password*. Daftar *host* yang diizinkan mengakses layanan RSH disimpan pada *file* `/home/.rhosts` dan `/etc/hosts.equiv`. *File* `.rhosts` dapat diisi oleh IP *address* maupun berupa *hostname* (diambil dari `/etc/hosts`). Pada setiap *node* di dalam *cluster*, *file* `.rhosts` harus berisi IP *address* dari seluruh *ethernet card* yang dimiliki oleh *head node*. Berikut langkah-langkah konfigurasi rsh-*server* pada Linux OpenSuSE 11.1 :

1. Masuk ke YaST | *Software Management*

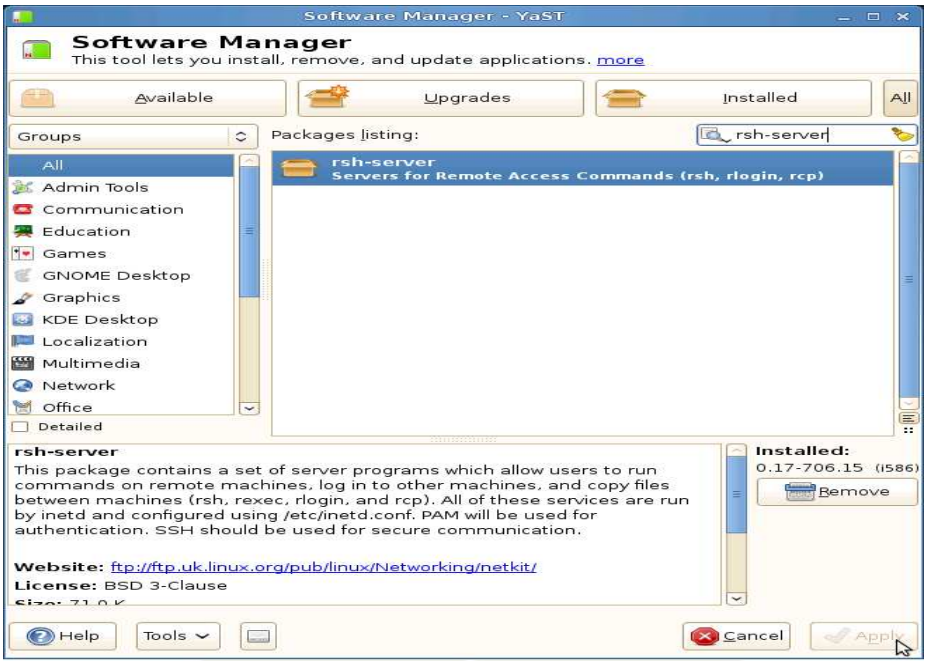
Gambar 3.7 Menu YaST *Software Management*

2. Pada kolom *Search*, diketik rsh dan rsh-*server*.



Gambar 3.8 Halaman *Software Manager*

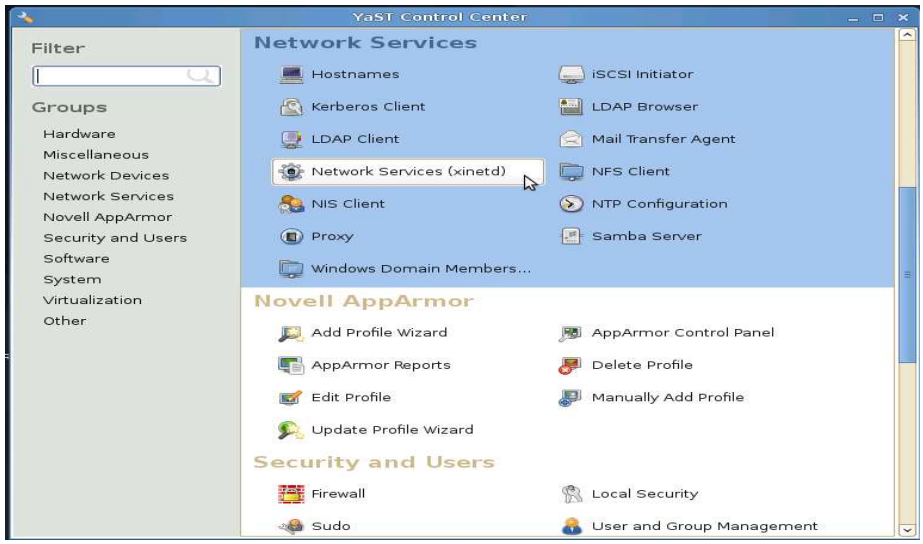
3. Dipilih rsh dan rsh-*server*, kemudian dipilih *install* | *apply*.



Gambar 3.9 Instalasi rsh dan rsh-*server*

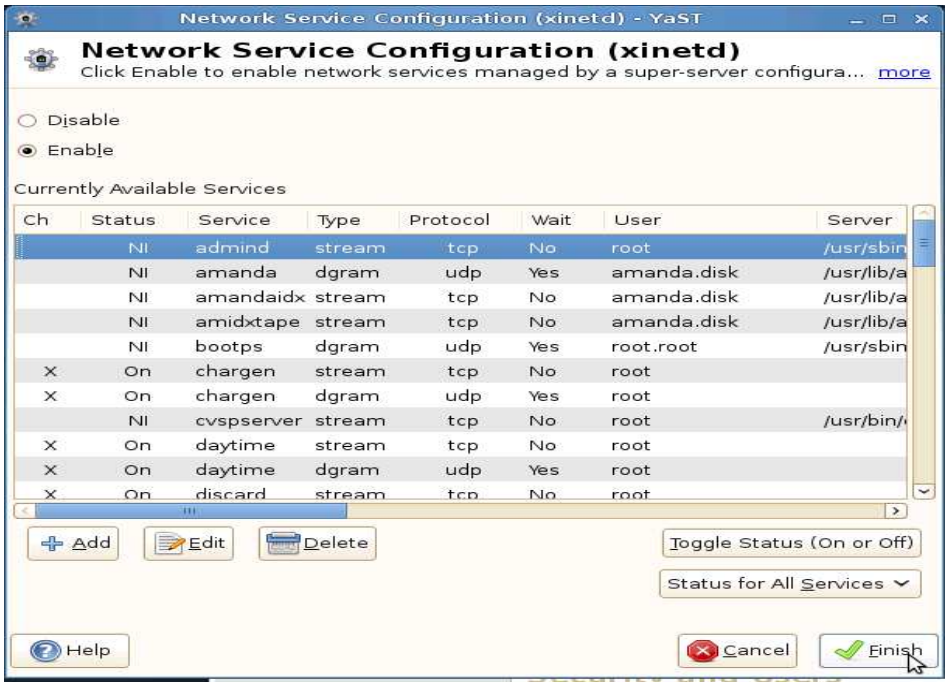
Setelah instalasi rsh-*server* selesai, perlu dilakukan konfigurasi layanan jaringan. Konfigurasi ini dilakukan untuk mengaktifkan layanan rlogin dan *remote shell* (RSH). Adapun langkah-langkah konfigurasi layanan jaringan adalah sebagai berikut:

1. Masuk ke YaST | *Network Services* (xinetd)



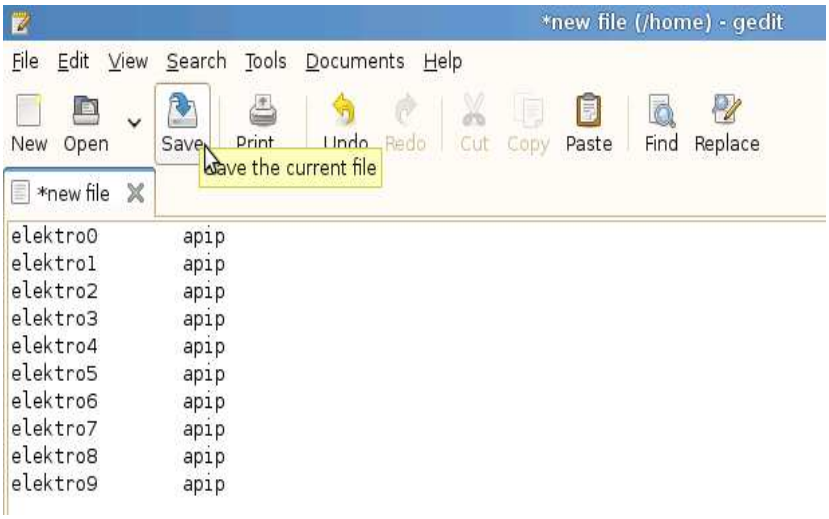
Gambar 3.10 Menu YaST *Network Services* (xinetd)

2. Dipilih *Enable* | *Finish*



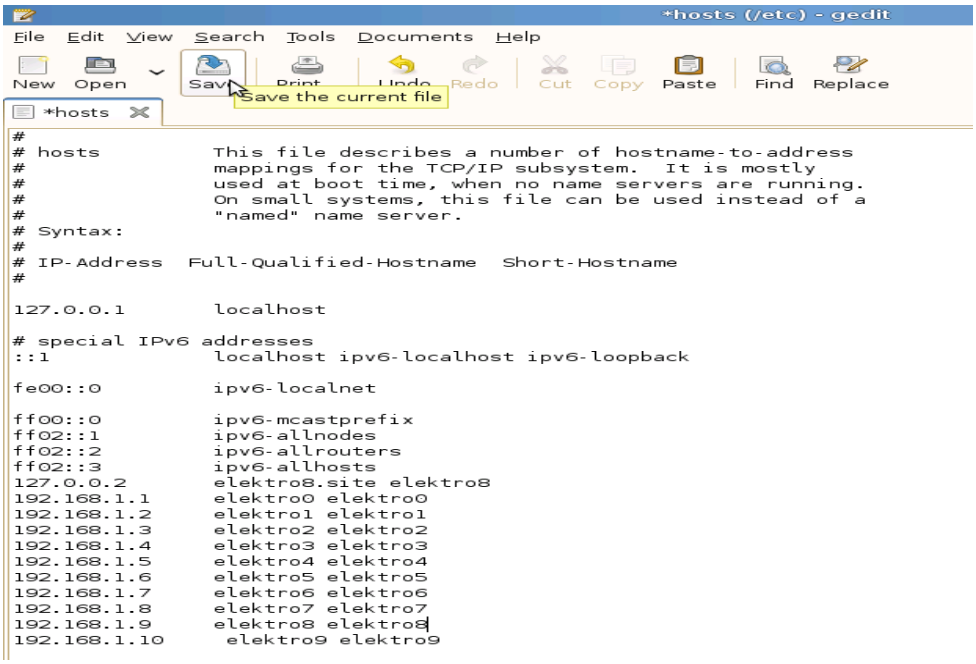
Gambar 3.11 Konfigurasi *Network Services*

Setelah rlogin dan rsh diaktifkan, langkah selanjutnya adalah pembuatan *file* .rhosts pada *directory* `/home/.rhosts` dan pengubahan *file* `/etc/hosts.equiv`. *File* ini berisi nama *hostname* dan *username* anggota *cluster* yang diizinkan mengakses layanan rsh. Untuk membuat *file* ini kita harus *log in* sebagai *root*.



Gambar 3.12 *File* .rhosts

Untuk daftar PC yang digunakan dalam *cluster* terletak pada *directory* `/etc/hosts`. *File* ini berisi IP *address* dan *hostname* dari masing-masing anggota *cluster*.



Gambar 3.13 *File hosts*

Selanjutnya menambahkan layanan rsh, rlogin dan rexec pada *file* `/etc/securetty`.

3.4 Instalasi dan Konfigurasi MPICH

Aplikasi MPICH harus diinstalasi ke dalam *cluster* untuk dapat digunakan. Proses instalasi dilakukan di setiap PC dengan *directory* instalasi diletakkan di `/usr/local`. Sebelum melakukan instalasi MPICH, ada beberapa prasyarat yang dibutuhkan. Antara lain:

1. *Copy*-an dari distribusi mpich.tar.gz
2. C *compiler*
3. C++ *compiler* untuk menuliskan program MPI.

Langkah-langkah instalasi MPICH pada komputer *head node* adalah sebagai berikut:

1. Diekstrak mpich.tar.gz

   ```
   tar zxvf mpich.tar.gz
   ```

2. Dilakukan konfigurasi dan menentukan dimana MPICH akan diinstalasi

   ```
   cd mpich-1.2.7p1
   ./configure –prefix=/usr/local
   make
   make install
   ```

Setelah MPICH selesai diinstalasi, langkah selanjutnya adalah menambahkan nama mesin pada *file* `machines.LINUX` yang terletak pada *directory* `/usr/local/share/machines.LINUX`. Untuk instalasi MPICH pada *compute node* 1 sampai dengan *compute node* 9, dapat dilakukan dengan cara yang sama.

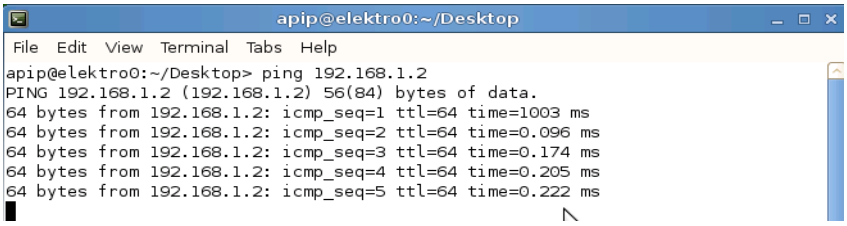
BAB IV

PENGUJIAN DAN PEMBAHASAN *CLUSTER*

4.1 Pengujian Koneksi Jaringan

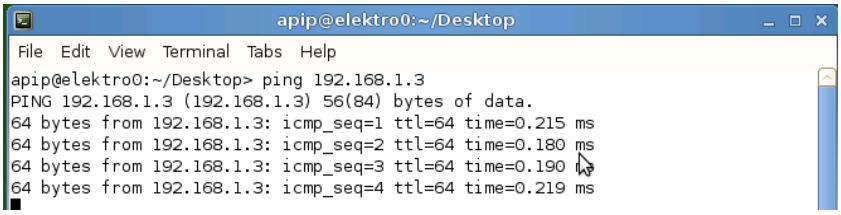
Setelah dilakukan konfigurasi, perlu dilakukan pengujian terhadap jaringan yang telah dibuat. Untuk pengujian koneksi jaringan, diklik kanan pada *desktop*, dipilih *open in terminal*, kemudian diketikkan perintah ping diikuti IP *Address* tujuan. Adapun hasil dari perintah ping dari *head node* ke setiap *compute node* adalah sebagai berikut:

1. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.2



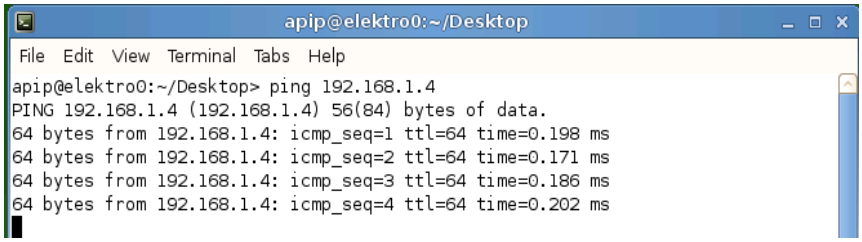
Gambar 4.1 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.2

2. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.3



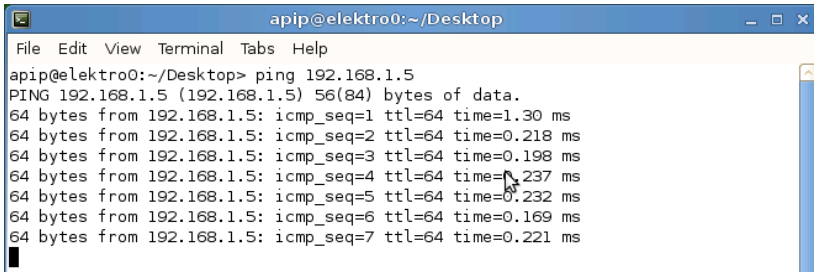
Gambar 4.2 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.3

3. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.4



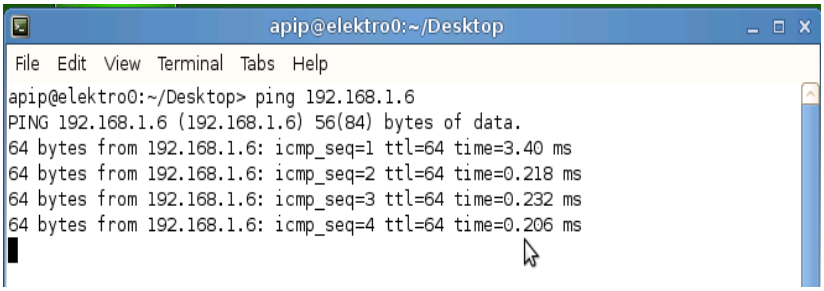
Gambar 4.3 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.4

4. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.5



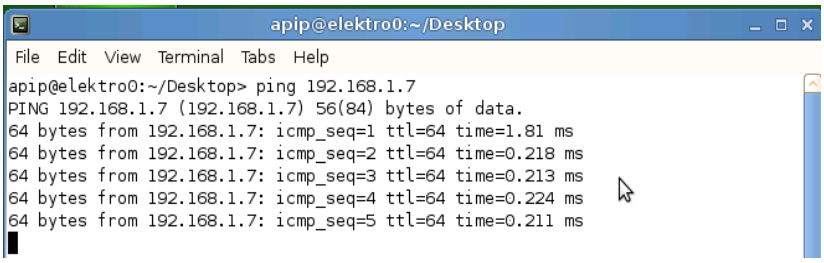
Gambar 4.4 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.5

5. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.6



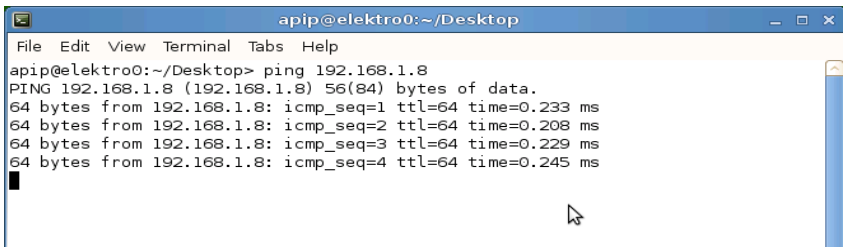
Gambar 4.5 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.6

6. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.7



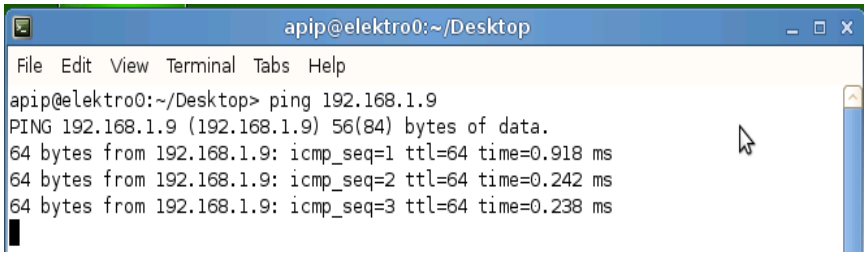
Gambar 4.6 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.7

7. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.8



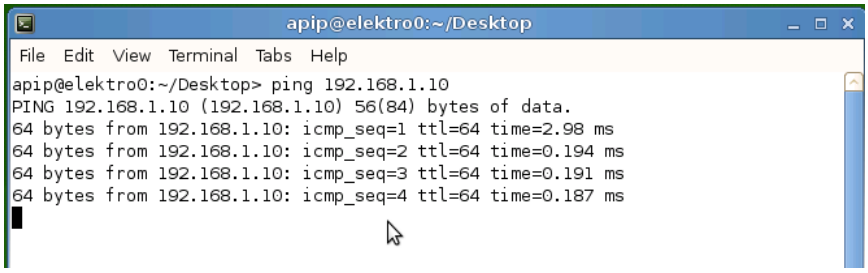
Gambar 4.7 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.8

8. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.9



Gambar 4.8 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.9

9. Ping dari IP *address* 192.168.1.1 ke IP *address* 192.168.1.10

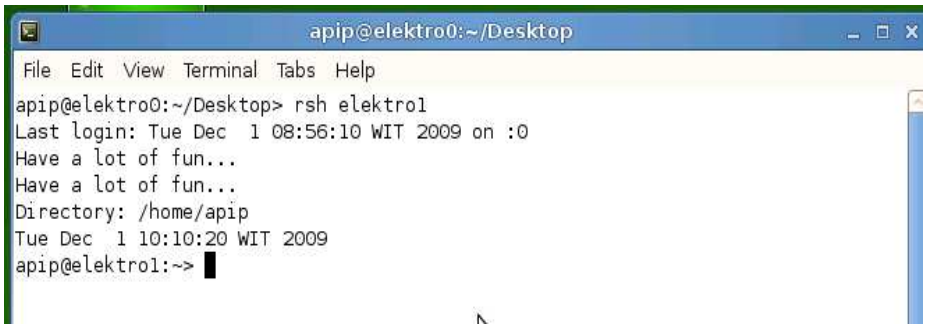


Gambar 4.9 Hasil Ping dari IP *Address* 192.168.1.1 ke IP *Address* 192.168.1.10

4.2 Pengujian *Remote Shell* (RSH)

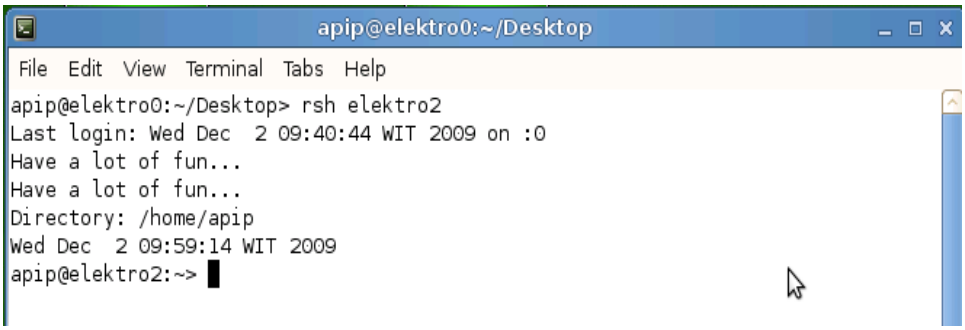
Dalam pengujian *Remote Shell* (RSH), perintah yang digunakan adalah "rsh" diikuti *namehost* dari komputer yang akan di-*remote.* Untuk pengujian *remote shell*, diklik kanan pada *desktop*, dipilih *open in terminal*. Adapun hasil dari *remote shell* dari *head node* ke setiap *compute node* adalah sebagai berikut:

1. *Remote shell* dari elektro0 ke elektro1



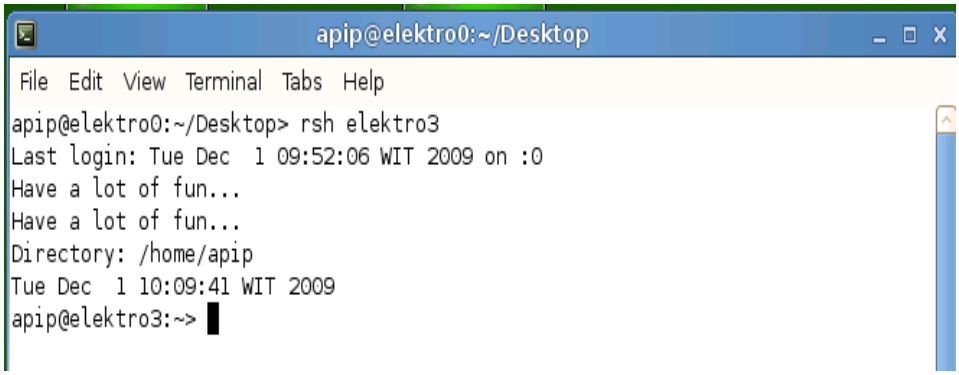
Gambar 4.10 Hasil *Remote* dari elektro0 ke elektro1

2. *Remote shell* dari elektro0 ke elektro2



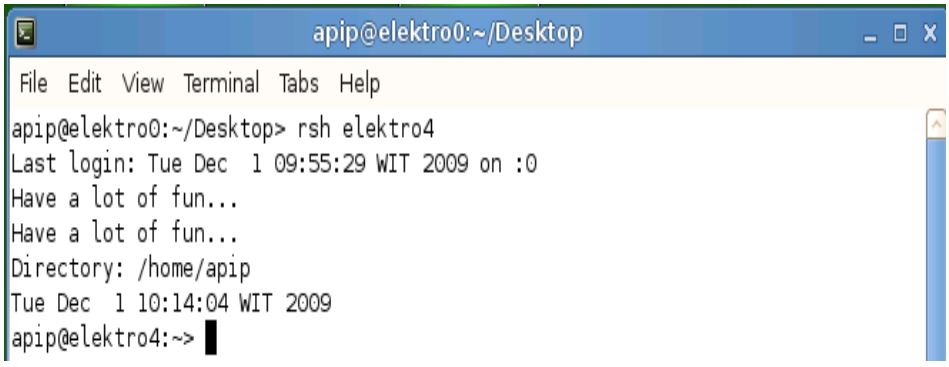
Gambar 4.11 Hasil *Remote* dari elektro0 ke elektro2

3. *Remote shell* dari elektro0 ke elektro3



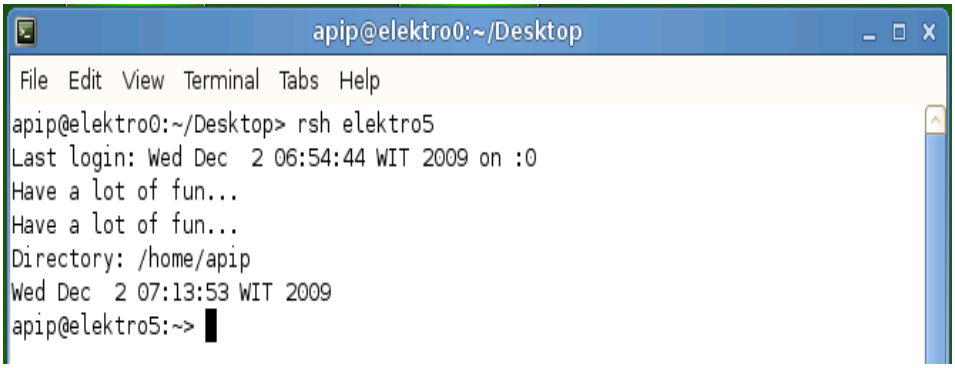
Gambar 4.12 Hasil *Remote* dari elektro0 ke elektro3

4. *Remote shell* dari elektro0 ke elektro4



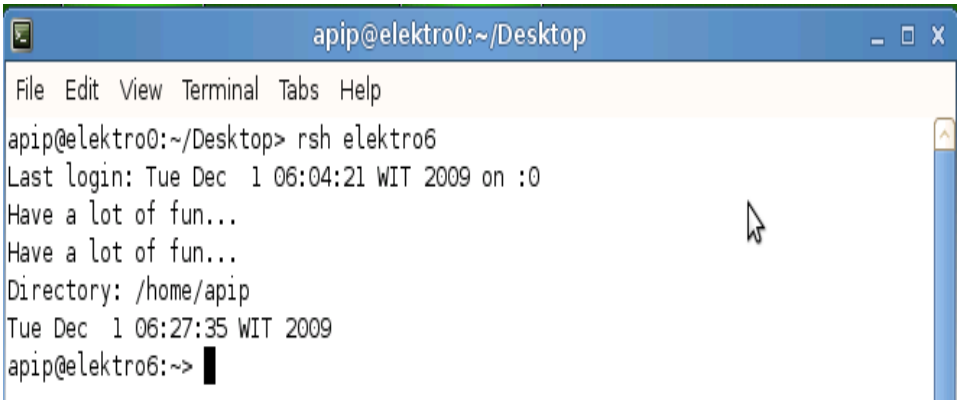
Gambar 4.13 Hasil *Remote* dari elektro0 ke elektro4

5. *Remote shell* dari elektro0 ke elektro5



Gambar 4.14 Hasil *Remote* dari elektro0 ke elektro5

6. *Remote shell* dari elektro0 ke elektro6

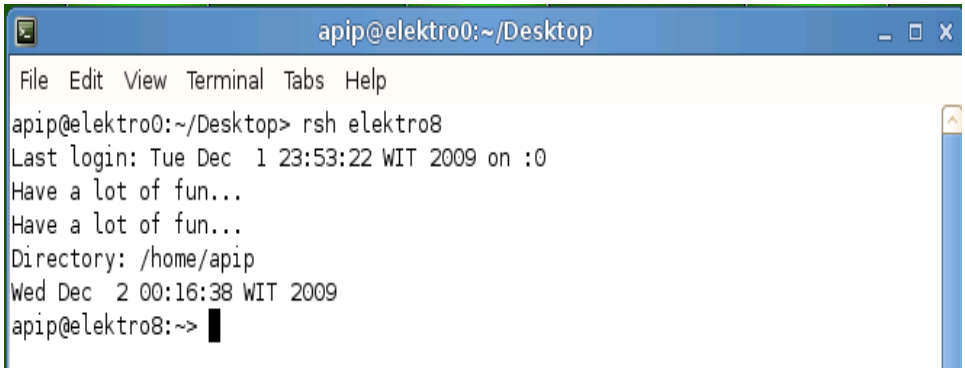


Gambar 4.15 Hasil *Remote* dari elektro0 ke elektro6

7. *Remote shell* dari elektro0 ke elektro7

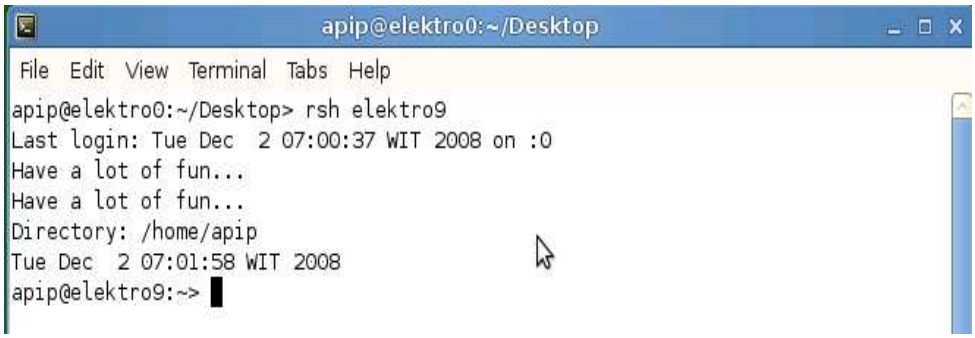
Gambar 4.16 Hasil *Remote* dari elektro0 ke elektro7

8. *Remote shell* dari elektro0 ke elektro8



Gambar 4.17 Hasil *Remote* dari elektro0 ke elektro8

9. *Remote shell* dari elektro0 ke elektro9



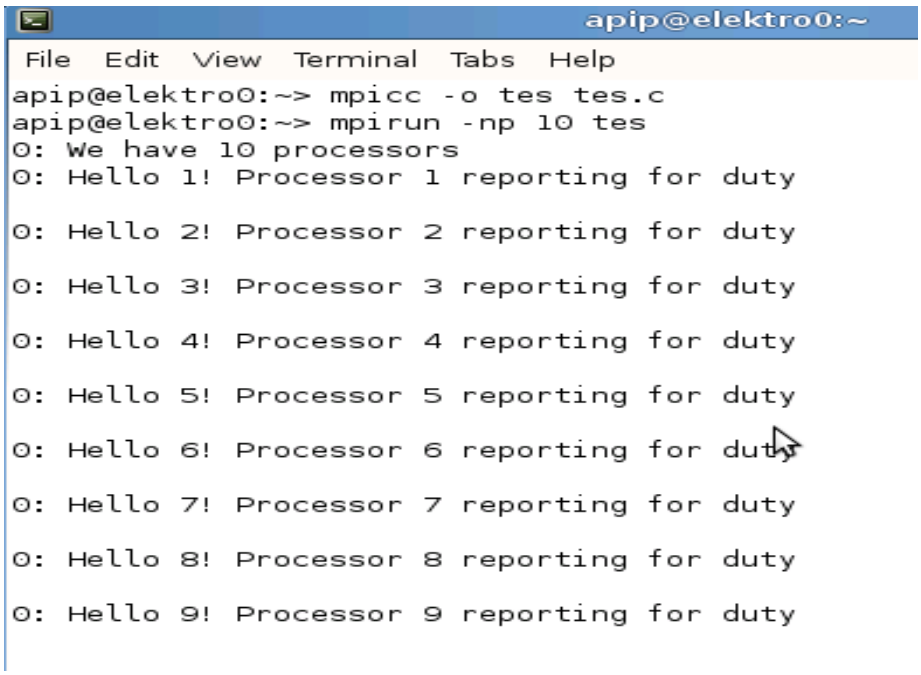
Gambar 4.18 Hasil *Remote* dari elektro0 ke elektro9

4.3 Pengujian MPICH

Pengujian MPICH (*Message Passing Interface Chameleon*) dilakukan untuk menguji apakah PC *cluster* yang dibangun dapat menjalankan program paralel. Pengujian program paralel menggunakan program tes.c, yaitu program yang dibuat untuk menampilkan jumlah *node* yang digunakan. Disini hanya untuk membuktikan apakah *node cluster* bisa menjalankan program secara bersamaan. Setelah di-*compile* dengan menggunkan perintah `mpicc -o tes tes.c`, akan dihasilkan *file* tes yang dapat dieksekusi.

Karena sistem ini merupakan sistem komputer terdistribusi maka *file* yang dijalankan harus sama dan berada pada *directory* yang sama pula. Pada percobaan ini *file* tes di-*copy*-kan ke semua komputer secara manual di-*directory*

`/home/apip`. Dengan menjalankan perintah `mpirun –np 10 tes`, maka akan muncul tampilan seperti gambar 4.19.



Gambar 4.19 Menjalankan Program Tes

Dari hasil penggunaan contoh program tersebut diatas terlihat bahwa semua anggota *cluster* dapat berkomunikasi. Sebagaimana diketahui, semakin banyak komputer akan membantu mempercepat proses permasalahan yang dikerjakan. Tetapi kita juga harus memperkirakan apakah permasalahan yang kita kerjakan ini efektif untuk dikerjakan oleh sistem komputer tunggal atau sistem *cluster*.

Bila pekerjaan itu relatif kecil atau ringan, maka sangat tidak efektif bila kita menggunakan sistem *cluster* karena diperlukan waktu untuk berkomunikasi dan mengkoordinasikan pekerjaan yang akan diselesaikan. Memerlukan waktu untuk membagi, mengirim, menerima, dan menyusun kembali permasalahan yang dikerjakan. Selain itu perlu diperhatikan pula topologi jaringan yang digunakan, apakah topologi yang dipilih mempercepat pengiriman data atau tidak.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini telah berhasil membangun PC *cluster* yang dapat menjalankan kompilasi pemrograman paralel pada *cluster* yang dibangun.

5.2 Saran

Untuk melakukan pengembangan lebih lanjut, beberapa hal yang dapat dilakukan adalah sebagai berikut :

1. Melakukan pengujian performa komputasi paralel untuk jumlah *node* yang lebih dari 10 unit.
2. Mengembangkan keamanan *cluster*.
3. Peningkatan *cluster* menjadi *grid* sehingga *cluster* dapat diintegrasikan dengan *cluster* yang lain.
4. Menambahkan aplikasi untuk menentukan *speedup* pada jaringan PC *cluster*.

DAFTAR PUSTAKA

Ajinagoro, Bagus Irawan. "*Aplikasi Sistem Paralel Menggunakan Prosesor Host 486 Berbasis Linux Debian*". Teknik Elektro Sekolah Tinggi Teknologi Telkom, Bandung. 2005.

Al Azam, Moh Noor. "*OpenSuSE*" [Online] Available http://wiki.klas.or.id/tiki-index.php?page=openSUSE, diakses 19 Agustus 2009.

Barney, Blaise. "*Message Passing Interface*" [Online] Available http://computing.llnl.gov/tutorials/mpi/, diakses 19 Agustus 2009.

Handoko, Dwi,. dan Tri Sampurno. "*Tips Praktis Membangun Komputer Linux Cluster*" [Online] Available http://komputasi.inn.bppt.go.id/tutorial/TIP_pembangunan_PC_Klaster.pdf, diakses 19 Agustus 2009.

Jamal, Ade,. dan Pandriya Sistha. "*Kinerja Komunikasi Data Kolektif Broadcast pada PC Cluster*". Risalah Lokakarya Komputasi dalam Sains dan Teknologi Nuklir XVII, hal. 435-444. Agustus 2006.

Kurniawan, Prima. "*Topologi Jaringan Komputer*" [Online] Available http://prima.kurniawan.students-blog.undip.ac.id/2009/07/19/topologi-jaringan-komputer/, diakses 19 Agustus 2009.

Novell, Inc. "*YaST*" [Online] Available http://en.opensuse.org/YaST/About, diakses 19 Agustus 2009.

Pahlevi, Said Mirza. "*Komputasi Grid dan Paralel*". Risalah Lokakarya Komputasi dalam Sains dan Teknologi Nuklir, hal. 15-24. Agustus 2008.

Rahmat, Akbar,. dkk. "*Implementasi Paralel Merge Sort Menggunakan MPICH2 dan Perbandingannya dengan Implementasi Sekuensial*". Fakultas Matematika dan Ilmu Pengetahuan Alam Institut Pertanian Bogor, Bogor. 2009.

Rianandra. "*Komputasi Paralel Penyelesaian Persamaan Poisson 1-D dengan Massage Passing Interface (MPI) pada Jaringan Komputer Berbasis Linux*". Jurusan Fisika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Sriwijaya, Palembang. 2007.

Soemantri, Maman,. dan Ari Darmariyadi. "*Penerapan Pengolahan Paralel Model Cluster sebagai WEB Server*". Teknologi Elektro. Vol. 6, No. 1, hal. 41-50. Juni 2007.

Sofyan, Ahmad. "*YaST di SuSE*" [Online] Available http://ftp.ui.edu/bebas/v01/DLL/ServerLinux/node40.html, diakses 19 Agustus 2009.

Suhartanto, Heru. "*Kajian Perangkat Bantu Komputasi Paralel pada Jaringan PC*", Makara Teknologi, Vol. 10, No. 2, hal. 72-81. November 2006.

Supriadi, Andi,. dan Dhani Garitna. "*Memilih Topologi Jaringan Hardware dalam Desain Sebuah Jaringan Komputer*". Informatika Pertanian. Vol. 16, No. 2, hal. 1037-1053. Februari 2007.

Widyaputra, Gentur. "*Perancangan Cluster Linux untuk Komputasi Paralel Octave*". Teknik Elektro Universitas Gajah Mada, Yogyakarta. 2008.

DAFTAR LAMPIRAN

Lampiran **Halaman**

DAFTAR TABEL

Tabel **Halaman**

DAFTAR GAMBAR

Gambar **Halaman**

DAFTAR SINGKATAN

ADI	:	*Abstract Device Interface*
API	:	*Application Programming Interface*
CPU	:	*Central Processing Unit*
IP	:	*Internet Protocol*
MPI	:	*Message Passing Interface*
MPICH	:	*Message Passing Interface Chameleon*
MPP	:	*Massive Parallel Processing*
NIC	:	*Network Interface Card*
PC	:	*Personal Computer*
PVM	:	*Parallel Virtual Machine*
RSH	:	*Remote Shell*
SLED	:	*SuSE Linux Enterprise Desktop*
SLES	:	*SuSE Linux Enterprise Server*
SMP	:	*Symmetric Multi Processing*
SSH	:	*Secure Shell*
SuSE	:	*Software und System Entwicklung*
YaST	:	*Yet another System Tools*

DAFTAR RIWAYAT HIDUP

Afif Zulfikar, lahir di Air Molek 20 April 1987 sebagai anak 3 dari 3 bersaudara. Menamatkan pendidikan sekolah dasar di SD Negeri 002 Rengat sekarang SDN 006 Rengat dari tahun 1993-1999. Pada tahun 1999-2002 menjadi siswa di SLTP Negeri 3 Air Molek sekarang SMPN 2 Air Molek. Pada Tahun 2002-2005 melanjutkan sekolah di SMK Negeri 5 Pekanbaru pada Jurusan Teknik Elektronika Komunikasi.

Pada tahun 2005, melanjutkan pendidikan ke Universitas Islam Negeri Sultan Syarif Kasim Riau Pekanbaru. Penulis Mengambil Jurusan Teknik Elektro pada Fakultas Sains dan Teknologi. Untuk memperoleh gelar sarjana, penulis telah melakukan penelitian Tugas Akhir dengan judul ***"Membangun PC Cluster di Lingkungan Laboratorium Komputer Teknik Elektro UIN SUSKA Riau".*** Penulis dapat dihubungi di e-mail: apip.666@gmail.com, dan YM di apip_666_x@yahoo.co.id.

Lampiran A
Kode Program Tes.c

```
#include <mpi.h>
 #include <stdio.h>
 #include <string.h>

 #define BUFSIZE 128
 #define TAG 0

 int main(int argc, char *argv[])
 {
  char idstr[32];
  char buff[BUFSIZE];
  int numprocs;
  int myid;
  int i;
  MPI_Status stat;

  MPI_Init(&argc,&argv); /* all MPI programs start with MPI_Init; all 'N'
  processes exist thereafter */
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs); /* find out how big the
  SPMD world is */
  MPI_Comm_rank(MPI_COMM_WORLD,&myid); /* and this processes' rank
  is */

  /* At this point, all the programs are running equivalently, the rank is used
     distinguish the roles of the programs in the SPMD model, with rank 0 often
     used specially... */
  if(myid == 0)
  {
    printf("%d: We have %d processors\n", myid, numprocs);
    for(i=1;i<numprocs;i++)
    {
      sprintf(buff, "Hello %d! ", i);
      MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
    }
    for(i=1;i<numprocs;i++)
    {
```

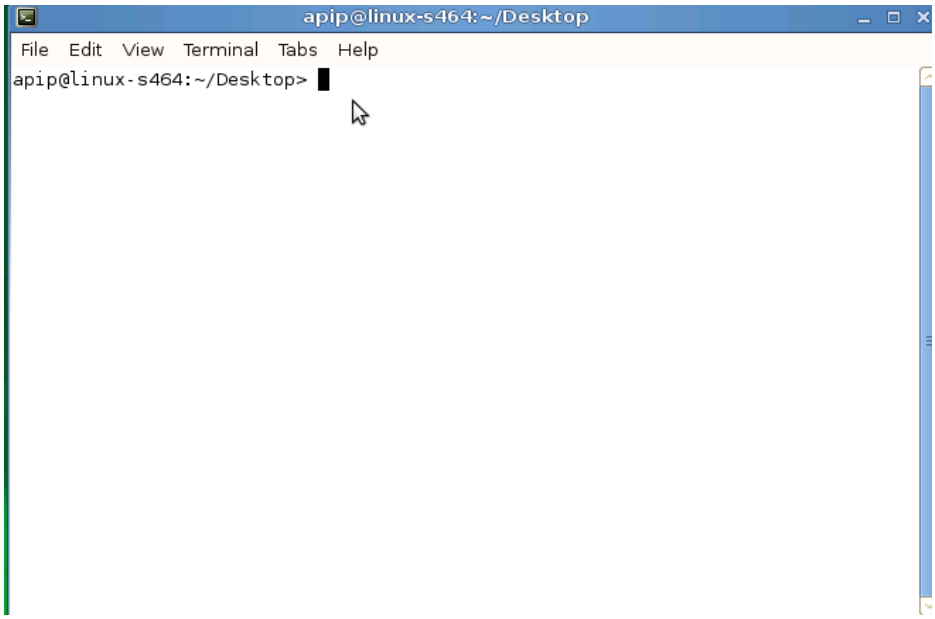
```
      MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD,
&stat);
      printf("%d: %s\n", myid, buff);
    }
  }
  else
  {
    /* receive from rank 0: */
    MPI_Recv(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD,
&stat);
    sprintf(idstr, "Processor %d ", myid);
    strcat(buff, idstr);
    strcat(buff, "reporting for duty\n");
    /* send to rank 0: */
    MPI_Send(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD);
  }

  MPI_Finalize(); /* MPI Programs end with MPI Finalize; this is a weak
synchronization point */
  return 0;
 }
```

Lampiran B
Biaya Pembangunan PC *Cluster*

No	Kegiatan/ Barang	Jumlah	Biaya Satuan	Total Biaya
1	*Switch* 8-*port* TP-Link TL-SF1008D	2 buah	Rp 150000	RP. 300000
2	Kabel UTP	6 meter	Rp 2000	Rp. 12000
3	*Connector* RJ 45	4 buah	Rp 1000	Rp. 4000
Jumlah				Rp. 316000

Lampiran C
Manual PC *Cluster*

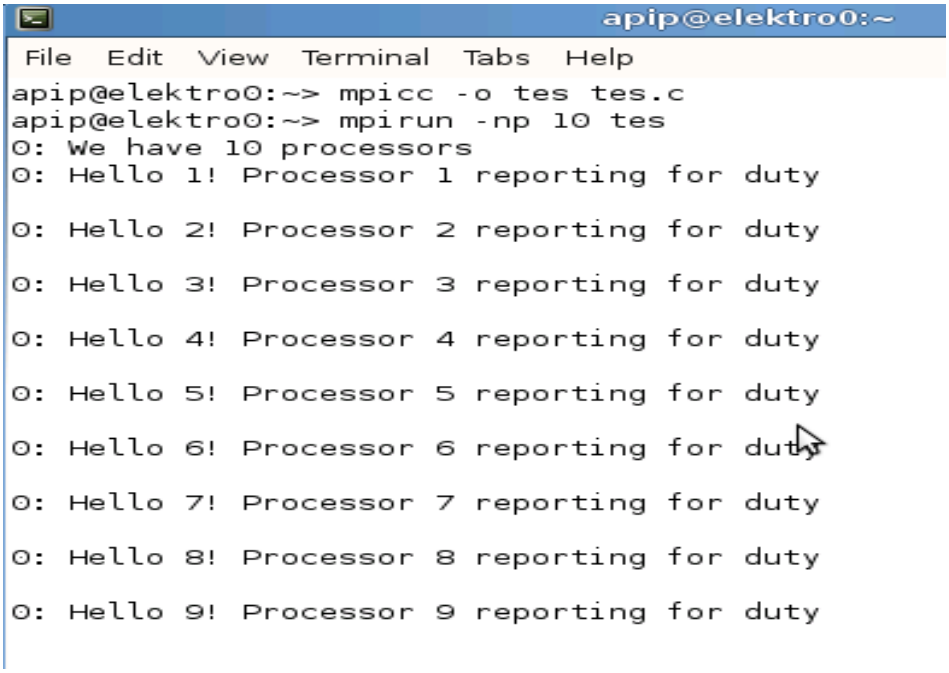
1. Memastikan setiap *copy*-an dari program yang dijalankan berada pada *directory* yang sama pada setiap *node*. Pada penelitian ini program yang akan dijalankan berada pada *directory* `home/apip`, dengan nama *file* tes.c.
2. Selanjutnya pada komputer *head node* diklik kanan pada *desktop* dan dipilih *open in terminal*. Akan tampil halaman terminal berikut.



Gambar 1. Halaman Terminal

3. Diketik cd untuk berpindah ke *directory* `home/apip`. Kemudian diketik perintah `mpicc –o tes tes.c` untuk meng-*compile file* tes.c.
4. Setelah *file* tes.c di-*compile* akan ada 2 buah *file* baru hasil *compile* yaitu tes.o dan tes. Hasil *compile* tersebut harus di-*copy*-kan ke setiap *node* pada *directory* `home/apip`.

5. Selanjutnya pada komputer *head node* diketikkan perintah `mpirun -np 10 tes`, maka akan muncul tampilan seperti gambar 2.



Gambar 2. Menjalankan Program Tes